Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

# TREBALL FINAL DE GRAU

**TÍTOL DEL TFG: Electrical Power Subsystem Functional and Environmental Testing for the 3Cat-4 mission**

**TITULACIÓ: Grau en Enginyeria d'Aeronavegació**

**AUTOR: Juan José Medina Musellas**

**DIRECTOR: Hyuk Park**

**TUTOR: Lara Fernández**

**DATA: 2 de septiembre del 2020**

**Título:** Electrical Power Subsystem Functional and Environmental Testing for the 3Cat-4 mission

**Autor:** Juan José Medina Musellas

**Director:** Hyuk Park

**Tutor:** Lara Fernández

**Fecha:** 2 de septiembre del 2020

**Resumen**

En este documento se explica de forma detallada que conocimientos son importantes para desarrollar un estudio ambiental y funcional de la Electrical Power Supply (EPS) P31u V8, y se puede extrapolar a otros dispositivos parecidos. Además, se detallan los resultados y conclusiones extraídos.

En un principio se introduce información sobre los CubeSats, la misión hacia la cual va dirigido el trabajo y el test de verificación que se llevará a cabo. Posteriormente, se dan a conocer los conocimientos básicos que son necesarios saber para el entendimiento del trabajo, aunque también es importante tener un conocimiento medio sobre electricidad y electrónica.

Posteriormente se explica cómo llevar a cabo la configuración del subsistema y de los diferentes test para, por último, hacer el test funcional y medioambiental en una cámara de vacío térmica (TVAC) y extraer resultados.

Gran parte de los procedimientos y especificaciones del test de TVAC, así como sus resultados, se pueden encontrar en el anexo. Los apartados expuestos en el cuerpo del documento son un resumen que contiene la información que se considera más importante, pero en los anexos la explicación es mucho más detallada y precisa. Los documentos presentados en los anexos y los diferentes tests han sido realizados por el autor y un compañero del laboratorio y también de la universidad, Albert Rodríguez, el cual ha ayudado mucho en transcurso del proyecto.

Por último, las conclusiones evalúan los resultados obtenidos anteriormente y deciden si el test ha sido exitoso o, por contrapartida, ha habido comportamientos de la EPS que no pueden ser aceptados y por ende el test ha fracasado.

**Title:** Electrical Power Subsystem Functional and Environmental Testing for the 3Cat-4 mission

**Author:** Juan José Medina Musellas

**Director:** Hyuk Park

**Tutor:** Lara Fernández

**Date:** September 2nd 2020

## Overview

This document explains in detail what knowledge is important in order to develop an environmental and functional study of the Electrical Power Supply (EPS) P31u V8, and it can be applied to the other similar power system devices. Also, the results and extracted conclusions of the test are exposed.

Initially, it introduces information on the CubeSats, the mission which the document is based on and the verification test that will be performed. Subsequently, the basic knowledge of the work is disclosed, although it is important to have a medium knowledge about electricity and electronics as well.

Afterwards, it explains how to carry out the configuration of the subsystem and the different tests to, finally, perform the functional and environmental test in a thermal vacuum chamber (TVAC) and extract results.

Much of the procedures and specifications of the TVAC test, as well as its results, can be found in the annex. The sections in the body of the document are a summary that contains the information that is considered most important, but in the annexes the explanation is much detailed. The documents presented in the annexes and the different tests have been made by the author and a colleague from the laboratory and also from the University, Albert Rodríguez, who has helped a lot during the project.

Finally, the conclusions evaluate the results obtained previously and decide whether the test has been successful or, on the other hand, there have been behaviors of the EPS that cannot be accepted which therefore made the test fail.

# INDEX

# LIST OF FIGURES

# ABBREVIATIONS

| | |
|---|---|
| ADCS | Attitude Determination Control System |
| COMMS | Communications Management System |
| EPS | Electrical Power System |
| GND | (Electrical) Ground |
| GOSH | GomSpace Shell |
| MTQ | Magnetorquers |
| OBC | On Board Computer |
| PCB | Printed Circuit Board |
| PFM | Proto-Flight Model |
| PoL | Point of Load |
| RBF | Remove Before Flight |
| TRP | Temperature Reference Point |
| TRPT | Test Report |
| TSTP | Test Specifications and Test Procedures |
| TVAC | Thermal Vacuum Chamber |
| UHF | Ultra-High Frequencies |

# 1. INTRODUCTION

The objective of this document is to verify the correct functionality of the Electrical Power Supply (EPS) under different environmental conditions. The document aims to explain all the process followed to carry out the tests in a concise and direct manner, but there is a more in detail explanation of the test procedures and results in the Annex (Section 8).

In this section, a brief overview of the CubeSats history and $^{3}$Cat-4 Mission will be presented in order to understand the context of this project. Also, there is an introduction to the Test Verifications in order to be aware of the importance of the Thermal Vacuum Chamber (TVAC) test that will be performed. The TVAC is a machine used for testing devices in space-like conditions. It depressurizes the internal chamber extracting the air and subjects it to the desired temperatures (within the feasible temperature range that the TVAC machine can provide).

## 1.1. CubeSat's History

The CubeSats arose from a University project when both Jordi Puig-Suari of California Polytechnic State University and Bob Twiggs of Stanford University proposed to their students building a spacecraft of reduced dimensions, but similar in functionality to the first Sputnik. The typical dimensions are a cube with side of 10 cm and few kilograms.

It was not intended to create a standard, but it became one because companies realized the advantages of producing this kind of satellite. Even though these satellites cannot substitute the typical satellites, they can complement them, especially for research activities.

The development and launch of one satellite can cost a few hundred million USD$, while for the CubeSats case it can cost a few hundred thousand USD$. Moreover, a great part of this amount is invested in the rocket, the one which will launch the satellite/s.

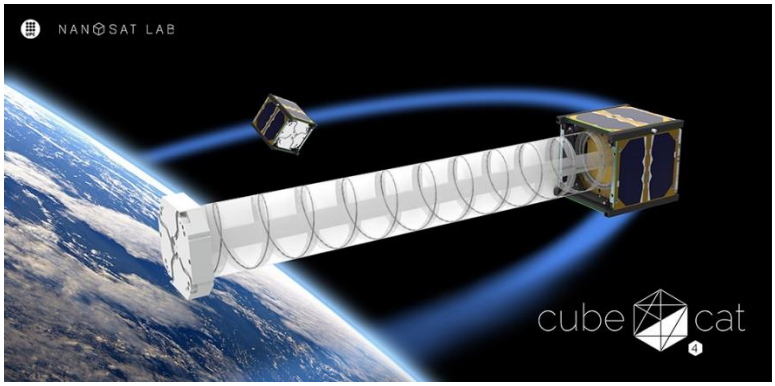



*Figure 1.1 - Artistic view of CubeCat4,NanoSatLab.*

*Retrieved from:* *https://nanosatlab.upc.edu/en/shared/images/3cat4_render.jpg/image_view_fullscreen*

In the case of the typical satellites the rocket can carry up to 2 or 3, while in the CubeSats case it can carry a huge number, much higher. In order to put some

numbers, at the beginning of 2017 the Indian's official space agency (ISRO) launched 104 satellites in one rocket.

This fact allows these satellites to perform missions that otherwise would not be performed because of the large budget required.

## 1.2. $^{3}$Cat-4 Mission

The 3Cat-4 (read "cube-cat-four") is the fourth member of the CubeSat series of UPC's NanoSat Lab. The main objective of this mission is to introduce a group of students in the technologies and methodologies which are currently used in space missions. Also, the mission has a lot of scientific and technological purposes.

Europe was leading the field of Global Navigation Surveillance System Reflectometry (GNSS-R) years ago, but the lack of economic resources in investigation, long-delays, etc. have made possible the NASA to take the lead.

Anyway, most of the knowledge in this field remains in Europe and the NASA feeds from it. Now, this mission can return the lead to Europe thanks to the CubeSats which have features that make them a good choice for this type of missions, as for example, the economic budget, which is much cheaper than the big satellite ones. So, the goal of the mission is to show the capabilities of the nanosatellites, the 1U CubeSat standard in specific, for the Earth observation using GNSS-R and L-band microwave radiometry, apart from helping the Automatic Identification System (AIS).

The main payload is called Flexible Microwave Payload (FMPL) including a GNSS-R reflectometer, which collects the data reflected from the Earth to evaluate different properties. It also includes an L-band radiometer which is used to estimate the brightness temperature (the temperature that would have a radiating body if it was a blackbody, i.e. emissivity equal to 1, and if the atmosphere had no effect on the radiance value) of a region determined by the

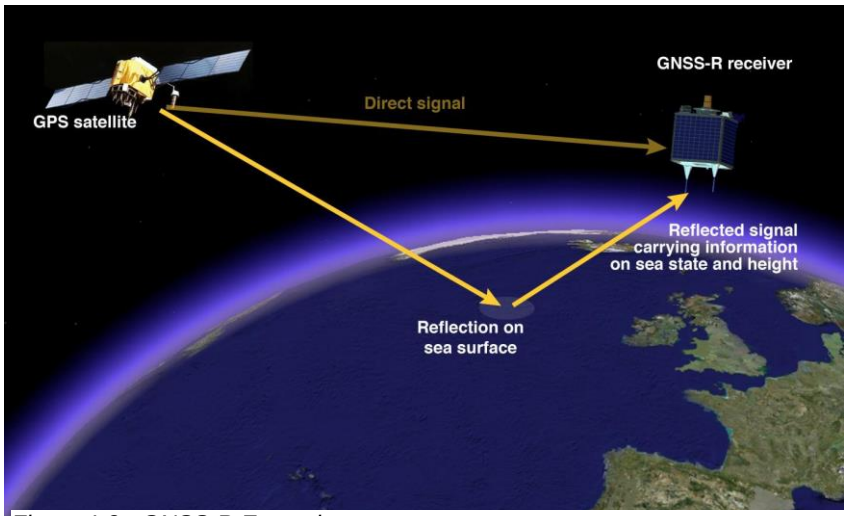



*Figure 1.2 - GNSS-R Example*
*Retrieved from:*
*https://www.eurekalert.org/multimedia/pub/web/109541_web.jpg*

antenna footprint. Lastly, it incorporates the AIS receiver which aims to help the current system of vessels monitorization gathering data from those which follow the AIS protocol.

The FMPL is based on a miniature Software Defined Radio (SDR) and it has its own dedicated On Board Computer (OBC) in order to process and gather all the data.

The satellite modes for the mission are the standard ones. The phases are: Launch and Early Operation Phase (LEOP), In-orbit commissioning Phase, Operations Phase and Decommissioning Phase.

In the LEOP phase, the satellite enters the Start-up mode after releasing the different three kill switches. Afterwards, in the In-orbit Commissioning Phase, the satellite detumbling mode is activated, which tries to reduce as much as possible the angular velocity and deploys the helix antenna. Moreover, the OBC carries out a health test to know if all the subsystems are working as expected, if this fails three times, the system is disconnected.

Once the health tests are completed, the satellite enters the sun safe mode, and if something unexpected happens, as for example, the loose of the nadir pointing, a high power consumption of a subsystem, a failure of an ADCS (Attitude Determination Control System) sensor, or a communications failure, the system goes automatically from Sun Safe into Survival mode. If the voltage underpasses certain threshold, the batteries will enter critical mode themselves and disconnect all Points of Load (PoLs). The PoLs are some pins of the EPS stack connector (H-47 to H-52) seen in the Annex (Section 8.1). The EPS has an internal system that can connect and disconnect each of them in order to control which subsystems of the satellite are enable/disabled.

If everything goes fine, the system will enter the third mission phase, the Operations Phase, in which it will gather all the data, which actually is the aim of the mission. The FMPL will not always be switched on, it will connect and disconnect following its duty cycle (around 5-10%).

Eventually, in the fourth phase (in about 1 year and 8 months if nothing unexpected happens), the satellite will reenter into the Earth’s atmosphere and disintegrate above 67 km from the Earth surface.

During all the mission, the EPS will play an especially important role because it is the subsystem in charge of supplying power to all the other subsystems, including the OBC. In order to check the subsystem requirements, the system must be verified and thus, undergo some tests.

## 1.3. Verification Tests

The verification tests are the ones which ensure that all the requirements which validate the subsystem for the mission are accomplished. Every requirement needs to be verified, and one of these requirements (and actually the one that this test wants to verify) is the temperature range.

There are different temperature ranges: the operational temperature range, the mission temperature range and the design temperature range. The operational temperature range is the temperature in which the manufacturer affirms the system can work as it is expected. The mission temperature range includes all the temperatures that the system is supposed to confront during the mission, this includes the worst of all the cases. The design temperature is the range in which the mission temperature range must be encapsulated. If the mission temperature range surpasses the design temperature range, the subsystem is not valid for the mission. Each institution has its own margins (although normally they are quite similar), so the document will center in the margins used in the NanoSatLab (the institution in which the test is carried out).

Usually, the design temperature range is +/-10 ºC of the operational temperature range, so if the operational temperature range is [-10,60] ºC, the design temperature range would be [0,50] ºC. The mission temperature range would have to be inside these temperatures.

If the mission temperatures agree with the design range, the subsystem will have to be tested. The first component design must pass the qualification level, in which the system is tested under temperatures of +/-10ºC with respect to the mission range. After the test of the component design has been successful, the next components build following the same design must only pass an acceptance test, which tests the subsystem under +/- 5ºC with respect to the mission range.

Finally, there is the protoflight test level, which is equal to the qualification level but the tested subsystem is flown in the mission after the test.

Normally, all these tests must be carried out a few times called cycles (as the temperature changes from hot to cold or vice versa, but always returns to de initial one). In the acceptance test there are 4 cycles, while in the qualification test there are 8 cycles. The protoflight test has the same level of demand as the qualification one, but there are only 4 cycles because it will be an onboard system and is better not to overstress it.

In this document, the test carried out will be an acceptance test. Nevertheless, the subsystem operational temperature will be also tested in acceptance level.

For the NanoSatLab, the acceptance level of the operational temperature range is +/-10ºC with respect to the operational temperature range itself.

## 1.4. Document goals

In this section the objectives of this document will be disclosed. As said before, the TFG is based in the GomSpace's P31U V8 EPS functional and environmental test, so the objectives will be divided in these two subgroups.

- Functional Test Objectives
    - Check the EPS Charging interface: Will succeed if the EPS can charge while connected and supplying energy to any PoL.
    - Check the housekeeping data extraction: Will succeed if the requested housekeeping data is returned in the defined structure and the values are correct.
    - Check that all the PoL can be enabled/disabled any time: Will succeed if all the PoL can be enabled/disabled when the user sends the command, within a short time frame (2 or 3 seconds at the most), and remain in this new state.
- Environmental Test Objectives
    - Check that the temperature of the EPS does not exceed or underpass the maximum and minimum temperatures in both charge and discharge mode: Will succeed if the temperatures of the sensors do not trespass these maximum and minimum temperatures.
    - Check all the Functional Test Objectives: Will succeed if all the Functional Test Objectives succeed during all the TVAC test.
    - Extract information about the voltage dependency with respect to the temperature: Will succeed if voltage and temperature data is extracted from the TVAC test.

# 2. STATE OF THE ART

The aim of this section is to compare the different technologies that are being used nowadays in the CubeSats, and concretely in the Electrical Power Subsystem.

The EPS could be divided into two modules:

- Power Control Circuit (PCC)

  The Power Control Circuit is the module which ensures battery charging and guarantees the electrical supply to all the other subsystems. This is the reason why the PCC will have to include diodes, microcontrollers and MPPTs to deal with its duties.

  In order to charge the batteries, there are two types of technologies: the MPPT (Maximum Power Point Tracker) and the PWM (Pulse Width Modulator). The PWM charges the battery at a voltage equal to the maximum voltage capacity, and when the battery is nearly charged, the PWM module sends pulses of variable width in order to charge them fully and smoothly without a sudden and sharp charge break, which damages the battery. This is not the most efficient method of charging, because the photovoltaic panels are not working in the maximum power point.

  On the other hand, the MPPT does exactly this, it makes the photovoltaic panel to work in its optimal point. This point will not have the same voltage as the battery you want to charge in most of the cases, so you will need a DC/DC converter in order to go from the photovoltaic panel voltage to the batteries one. If the DC/DC converter does not lose so much power, and the photovoltaic panels voltage is greater than the batteries one, the decrease in voltage will provoke an increase in current, and the batteries will charge at a higher rate.

- Batteries

  There are lots of battery types but they generally belong to two main groups. The primary cells or the secondary cells. The primary cells are batteries that cannot be reused after they have discharged because the chemical reaction is not reversible. The other ones can be recharged, and this is the main point that makes them the best option for a CubeSat power supply.

  Inside this group there are a lot of batteries made of different chemicals, but the ones that have the best performance, taking into account that the weight (and the price, hopefully) has to be minimized, are the Lithium batteries.

The Lithium batteries can be separated in two subgroups, the Lithium-Ion (Li-Ion) and the Lithium-Polymer (Li-Po). They both have pros and cons with respect to the other one.

The Li-Ion are batteries which have more power density, which means that they can give large amounts of energy in comparison of their size. In addition, they usually have greater voltage capacity than Li-Po batteries. On the contrary, these batteries are more dangerous than the Li-Po ones, because the electrolytical component is unstable and can combust when stressed under extreme temperatures or if hit or punctured. Although this information is true, this hardly ever happens in the battery normal use, it happens with the misuse of the battery.

So, in terms of safeness, the Li-Po batteries are the best choice. Moreover, the specific energy (energy stored per unit of mass) is better in the Li-Po because they are lighter than the Li-Ion. This means that for the same weight, the energy stored is larger in the Li-Po battery, but the volume occupied is larger too. The Li-Ion batteries optimize better the space while Li-Po ones optimize better the weight, with respect to the energy capacity.

Nevertheless, the Li-Ion batteries can charge faster and deliver more power, thanks to its higher power density, and their lifespan is greater than the Li-Po which makes them a good option for spacecrafts, which need fast charging and a life-span of various years. In addition, they are cheaper than the Li-Po batteries.

Eventually, I will compare the two different EPS architectures, the centralized and distributed ones. The centralized aims to use its buses for more than one module or subsystem connected to the EPS. For example, it could have a 3.3V and 5V regulated buses and every system would connect to its corresponding bus depending on the voltage needed.

The other approach is the distributed architecture, in which the bus is at a high voltage, in comparison to the centralized architecture, and every subsystem has to regulate the voltage to match the voltage needed. It is true that is less efficient because in order to regulate the voltage there will be losses in every subsystem, but the advantage is that this architecture can be reused in other missions with different specifications.

# 3. PREVIOUS STUDY

This section's objective is to introduce some topics that are particularly important in the course of the project. It is the previous study which I had to make before I could start the tests. It will be divided into two subsections, the Raspberry Setup and the I2C protocol.

## 3.1. Raspberry Pi basis and Setup

The first step in order to accomplish our objective was learning how Raspberry Pi and Raspbian (its operative system) work. Raspbian is a Linux distribution based on Debian and the functionality is remarkably similar to other Linux based OSs.

We had to flash one SD card with Raspbian OS and, as the visual interface is not necessary for the task, it is recommended to download the Lite version which is faster. The tool we used for flashing was the one called BalenaEtcher.
Once flashed, we had to communicate with the Raspberry Pi from one computer.

The simplest way is using SSH protocol, but it must be enabled in the Raspberry Pi's setup files. There are two ways to do it. The first one, is connecting the Raspberry Pi to a monitor or TV via HDMI, login with *username* and *password* (typically *pi* and *raspberry*, respectively), execute "sudo raspi-config" and enable the SSH interfacing option.

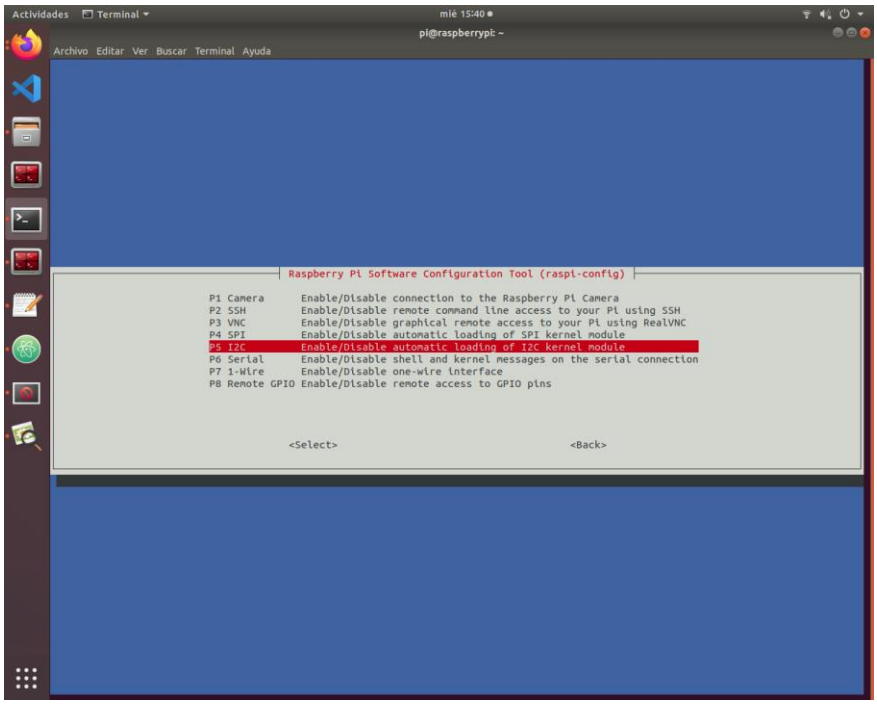



*Figure 3.1 - Raspberry's Configuration Tool*

For the second one, you must enter the boot partition of the flashed SD card and create an empty file called “ssh”.

Afterwards, you will be able to communicate with the Raspberry Pi via SSH protocol.

As we had some connectivity problems with the Raspberry Pi, I will describe 2 different setups for the communications between the Raspberry Pi and the computer:

- Direct Ethernet from computer to Raspberry Pi:

This is the simplest connection but hardest setup. We only needed one Ethernet cable like a RJ45.

The communication must be done in the same Network, so the Ethernet card of the computer and the ethernet card of the Raspberry Pi need to have same netmask.

We had to enter the rootfs partition of the SD card and open etc/dhcpcd.conf and uncomment and modify the following lines (except Example static IP Configuration).

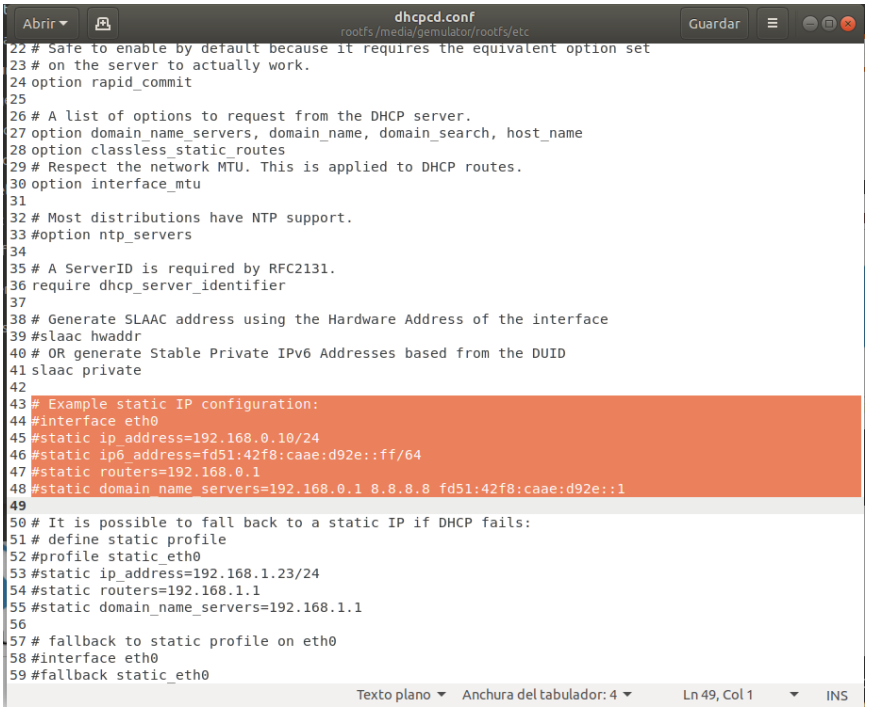



*Figure 3.2 - Raspberry’s Network Configuration*

We needed to change the static ip_adress field in order to match with the netmask of the computer's Ethernet card.

Finally, when the Raspberry had booted, we were able to connect with the Raspberry via ssh using the command "ssh pi@192.168.0.10".

The main problem of this configuration is that the Raspberry Pi will not have Internet connection, which is particularly useful at the beginning in order to download packets.
If your Raspberry Pi is close enough to a router, you could try to connect via Wi-Fi, but I do not recommend it because the Wi-Fi antenna only receives strong signals.

- Connection via Switch:

In this configuration the setup is quite easy, but there is more hardware involved. We needed a switch (or today's router, which has a switch) and two Ethernet cables. Hopefully, the switch (or router) will assign different IPs for both computer and Raspberry Pi via DHCP protocol.

Once assigned, we executed the command "nmap 10.4.33.0-255" for searching the new ip in that range. You must change it by your netmask in your nmap command. Usually the command will be like "nmap x.x.x.0-255" being "x" your netmask numbers.

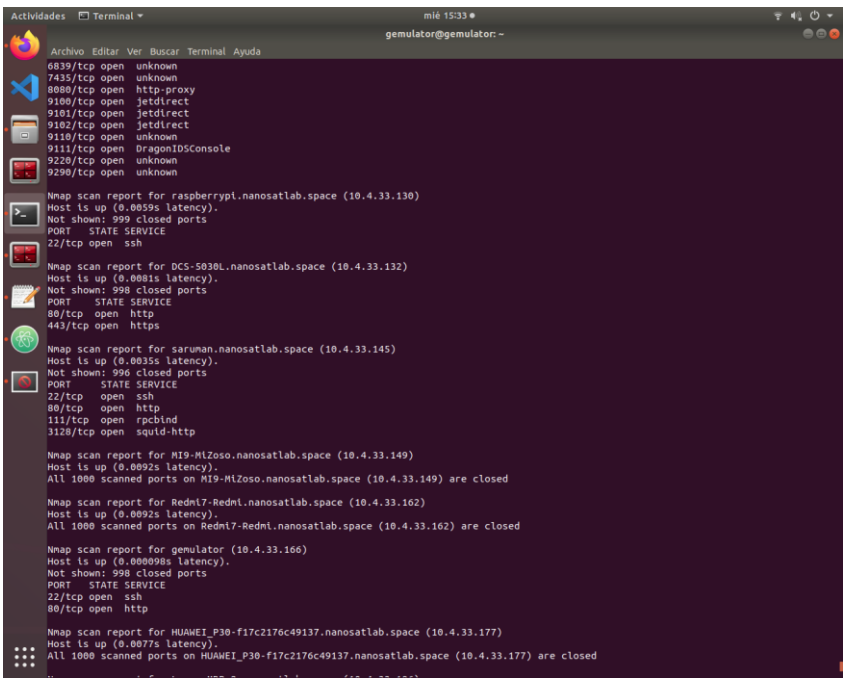



*Figure 3.3 - IP example*

Eventually, we executed the same ssh command, now changing the ip as “ssh pi@10.4.33.130”.

## 3.2. I2C

Another important topic is the I2C protocol. I2C is a two wired serial protocol for exchanging information bit by bit. These two wires are named SCL (serial clock) and SDA (serial data).

The SCL function is to synchronize the master (device who always starts the communication) with its slaves (device/s which will answer the request). On the other hand, the SDA is the medium in which the bits are exchanged.

The basis on the communication protocol will be explained down below, but if it is necessary you can refer to [1] for in detail explanation.

The I2C protocol starts leaving high the SDA while forcing down the SCL. At this point, the bus is controlled by the master, and no other master can interrupt the communication until it has finished the communication.

The first byte sent by the master is the slave’s I2C address, which permits the slave to know if the message is for him. It is a little endian 7-bit address, so there is one bit remaining, the read/write bit. This bit indicates whether you will be waiting response or not (Read 1, Write 0). After every byte is sent, there will be a 9$^{th}$ bit indicating if the slave has received the message (or know how to parse a response). Will be 0 if acknowledged or 1 otherwise.

After that, there are all the data bytes that you want to send.

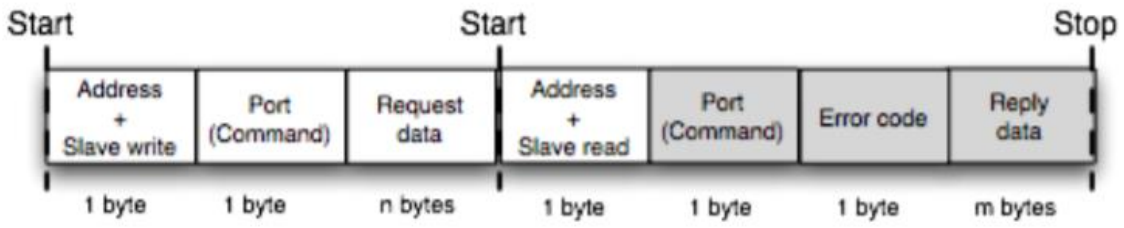



*Figure 3.4 - I2C EPS Command Sequence*

As we can see in Figure 3.4 the protocol that follows the EPS has a repeated start in order not to interrupt the communication and, moreover, do not let the other devices communicate. As the communication of the EPS is one master and one slave (the OBC, in this case the Raspberry Pi, is the master and the EPS is the slave) the repeated start is not necessary and can be replaced by a stop and start.

The EPS I2C protocol starts with the typical I2C of 7 bits address + write bit. After that, it has a Port byte. The Port specifies which command is going to execute the EPS, and all commands are specified in the EPS datasheet [2] (an example of these commands can be seen in the Annex Section 8.2).

Next, there is the request data byte, which is specified in each Port in order to complete the message.

Finally, there is a read message, with another 7 bits address header but now with a read bit. Following this header, the EPS will respond with the same Port byte and the data requested, which can have different lengths depending on which was the requested information.

# 4. METHODOLOGY

This section describes the steps followed to prepare and carry out the TVAC test. It is composed by the EPS setup, the EPS communication and the Environmental Test Preparation.

## 4.1. EPS SETUP

The EPS setup is divided into the Switch On section, the one that explains the procedure in order to successfully switch on the EPS, and the Configuration subsection, in which it is explained how the EPS must be debugged.

### 4.1.1. Switch On

For this purpose is needed a power supply, whether you want to turn on the EPS with batteries (because the batteries will need to charge) or bypass the batteries and provide the electrical power from the power supply.

In the next figure it can be seen the different connections on the top and bottom side of the EPS.

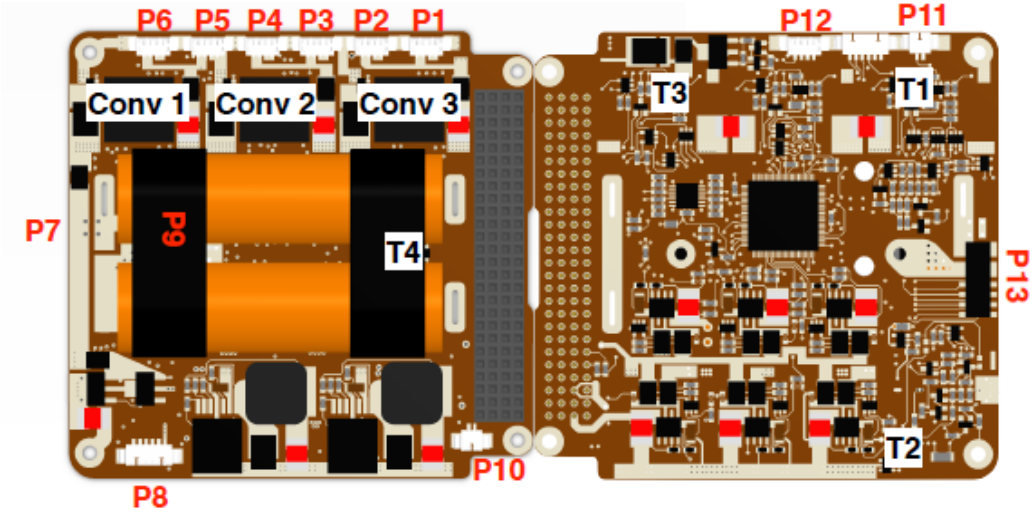



*Figure 4.1 - EPS Connections retrieved from EPS datasheet [2]*

Much of the P* connectors are picoblade connectors, so for the wiring you will need some picoblade crimp terminals and female housings, a crimping tool and some wires.

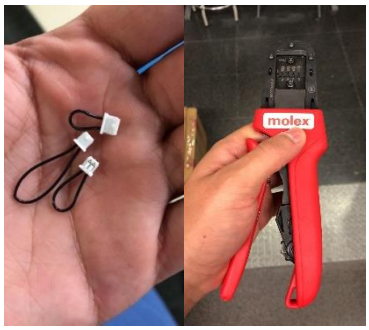



*Figure 4.2 - Example of picoblade wiring and crimping tool*

In order to switch on the EPS, the kill switch must be disabled. Kill switch is a safety measure which does not allow EPS to turn on even if an external power supply or the batteries are connected. For disabling the kill switch there are two options.

The first option is shorting the P8 (called Flight Panel) 3rd pin to ground, which will override the kill switch.

The other option is to interconnect both pins in each kill switch connector, i.e., P10 and P11.

Once this has been done, the system is ready to turn on. If you want to connect the batteries, you must short one or the two inner pins of the P7 connector (called Battery Arm Connector) to one or the two outer pins.

On the contrary, if you want to switch it on with an external power supply, you must bypass this connection by shorting $V^+$ (7.4V) to the previous outer pins and the electrical ground (GND) to any EPS ground to set the reference.

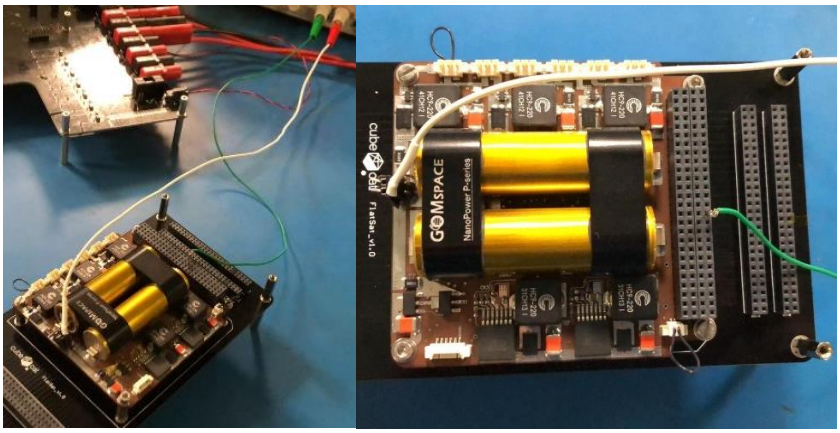



*Figure 4.3 - EPS Switch on Setup*

### 4.1.2. Configuration

After the switch on, the next step is the configuration of the EPS to stablish an I2C communication. In order to do that, we needed a FTDI with a 3.3V output voltage.

This cable allowed us to communicate with the EPS GOSH from one computer and change settings. We needed one program to debug the EPS (enter GOSH via FTDI). We used Minicom in an Ubuntu OS, but there is a program for Windows called RealTerm.

The connection of the FTDI to the computer is easy because the cable has a USB connector. On the other side is more complicated because you must attach different wires to the connections you are going to use. These wires (which end in the EPS) must be assembled into the P12 connector, so you will need all picoblade stuff mentioned in the past section.

| P12<br>Serial connector | |
|---|---|
| Pin | Usage |
| 1 | GND |
| 2 | Not connected |
| 3 | RxD |
| 4 | TxD |

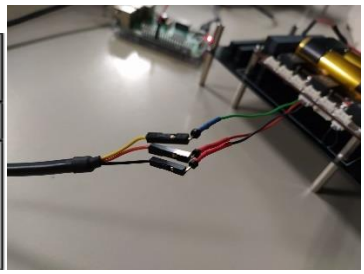


*Figure 4.4 - P12 Connection Scheme retrieved from EPS datasheet [2] and connection example*

Notice that the transmitter port must be shorted to the other device's receiver and vice versa.

In order to enter the GomSpace Shell (GOSH), I will describe the method we followed. We installed minicom "sudo apt install minicom" and then we used "sudo minicom -D /dev/ttyUSB2" in our case, but the number depends on the connected USB port. The baud rate was set to 500000 as specified in the EPS datasheet, so we did not have to change it.

If the connection is successful, once you press enter you must see the device name, "eps" in our case. After that, you are inside GOSH and you can change setting.

It is recommended to change the clock frequency of the EPS so that it matches with the clock frequency of the I2C master, otherwise the connection could fail.

Also, the EPS is configured to use CSP protocol, but we want the EPS to act as an I2C slave, so it is important to change it to I2C slave or “legacy mode”. For enabling it we used “board i2cslave 1”.
Another important information is the I2C slave direction of the EPS, which can be acquired by sending “board i2caddr”.

After this configuration has been made, the setup will be completed.

## 4.2. EPS COMMUNICATION

This section is centered in communication between our OBC, the Raspberry in this case, and the EPS. It explains the configuration that must be followed for the I2C bus to work correctly. Also, for the script part, it is explained what it did not work for us and the solution we gave to this problem

### 4.2.1. EPS Wiring Communication

Once we were connected to the Raspberry Pi remotely from our PC and the EPS was successfully turned on, we had to interconnect the Raspberry with the EPS. In the next figure, we can see the typical pinout of a Raspberry Pi.

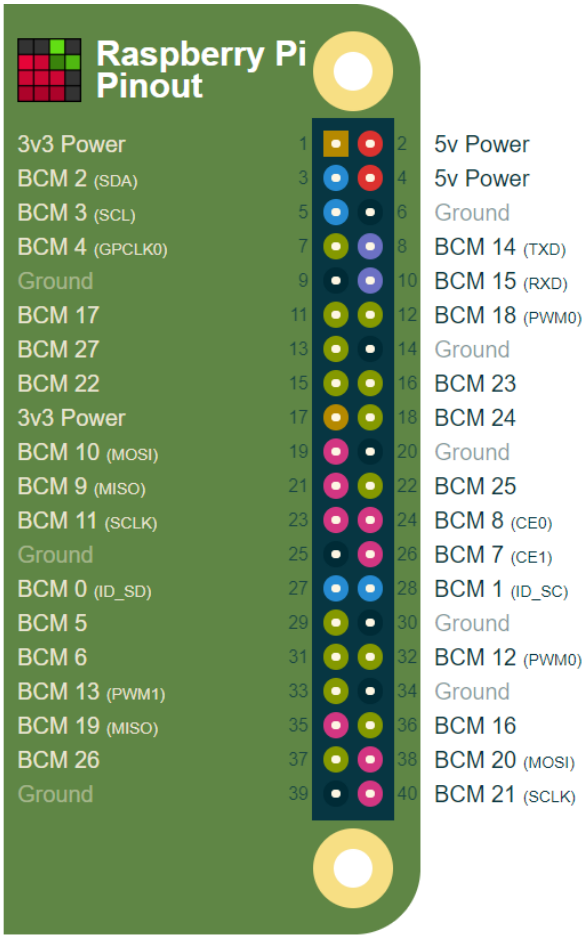



*Figure 4.5 - Raspberry Pi Pinout retrieved from [https://pinout.xyz/](https://pinout.xyz/)*

Following this scheme, only three pins are necessary. The pins 3, 5 (both I2C pins) and 39 (GND). Notice that pin 39 is a particularly important connection to do because if both grounds (EPS and Raspberry's one) are not interconnected, the reference will not be the same and the communication will irretrievably fail.

Raspberry's pins 3 and 5 must be shorted to EPS's H1-41 and H1-43 respectively. Raspberry's 39$^{th}$ pin must be shorted to any ground connector of the EPS. EPS stack connector is shown in Section 8.1.

### 4.2.2. C Script for EPS Communication

After the connections have been made, the communication protocol must be carried out. For this purpose, we initially tried an I2C python library called smbus2.

We made a script and looked in an oscilloscope what we were sending and if it corresponded to the EPS I2C protocol. The following figure is a reminder of the EPS I2C protocol so you can compare it with the smbus2 messages.

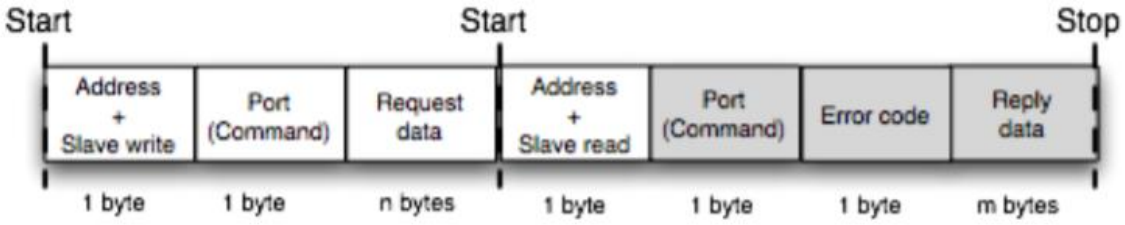



*Figure 4.6 - I2C EPS Command Sequence Reminder*

The two messages we sent were a write message to address 2 (the EPS I2C slave address), port 2 and one byte with all bits 1 and the other message with all them 0.

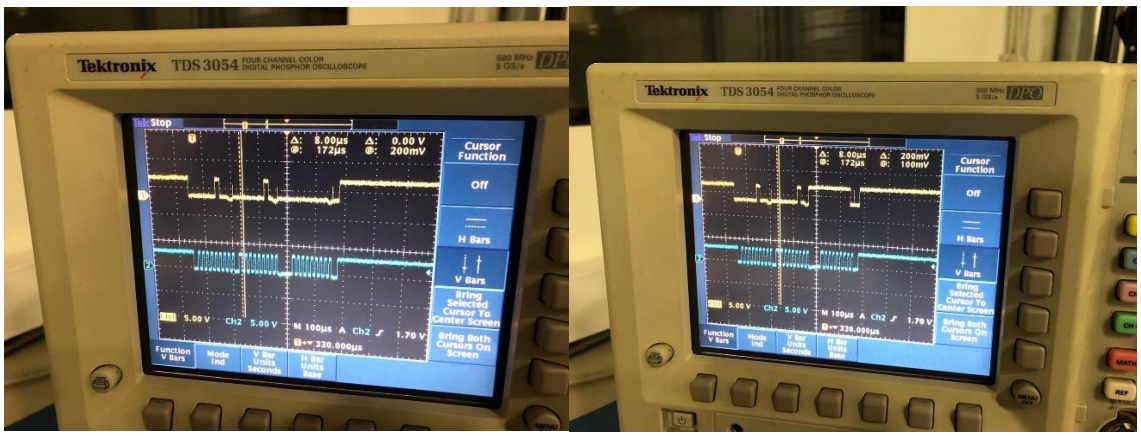



*Figure 4.7 - I2C Write Message Data Comparison*

As seen in Figure 4.7, the message scheme matches the EPS I2C protocol, but we wanted to ensure that the scheme was persistent even if there was a change in the port, so we changed from port 2 to 15, which is all bits 1.


*Figure 4.8 - I2C Write Message Port Comparison*

Here we can see that this change keeps following the EPS I2C protocol. The last thing remaining was trying the read message, but when we tried it, we discovered that it was not following the protocol.

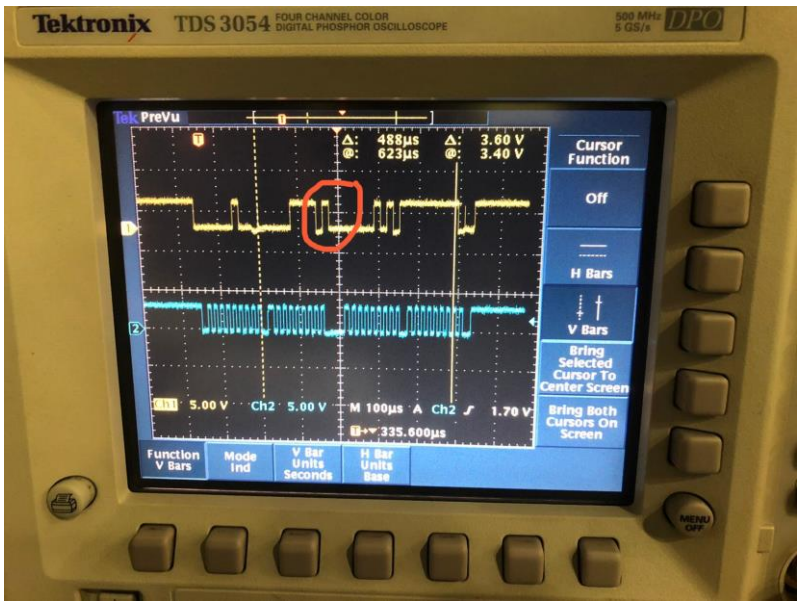



*Figure 4.9 - I2C Read Message*

Even the direction and port where right, the read/write bit must be in the final bit of the 1st byte, so the communication failed every time.

That is the reason why we changed to C language and used different libraries in order to create a valid script which made exactly what we wanted.

The C script is attached in the Annex (Section 8.3).
The script is composed by one struct declaration, 9 functions and the main.

The struct is declared as a union in order to map the bytes received to a struct containing all the data ordered by attributes. This struct is extracted from the EPS datasheet [2].

The functions are:

- eps_hk_print: This function shows all the data stored in the struct in a graphical view on the terminal.

- eps_hk_nth: This function allows the computer to map correctly all the bits received. The network byte order is big-endian, as the EPS byte order, so the C functions which convert from network byte order to host (our Raspberry's processor) byte order can be used to reorder these bytes.

- GomGetTelemetry: This function sends one write message and one read message for receiving all the housekeeping data in order to fulfill the struct. Notice that there is some sleep time between these two messages. This time is needed because the Raspberry opens one file (the I2C pin file) every time it needs to start one communication. If this communication is not closed yet, it will not be able to start a new one, and the overall communication will fail.

- SetOutputStatus: This function allows the EPS to supply power to the different PoLs. First, it uses the GomGetTelemetry to know which PoL are enabled and which of them are disabled. Then, it changes the PoL that you want to enable/disable. Finally, it asks for the telemetry another time, after some milliseconds have elapsed, in order to know if the PoL has actually changed.

- representTime: This function represents time and date in the Excel format in order to carry out the plots.

- file_log: This function generates a log file which contains all the important data extracted from the housekeeping. It is stored in a .csv file for its subsequent processing.

- customed_file_log: This function actually does the same as the previous one but changing the file name.

- SetOutput: This function gets information about the mission mode that you are trying to register as a parameter from the command line. Every time the OBC retrieves information from the EPS, it sets another time all the PoL to their corresponding state, regarding the mission mode. This had to be made because the PoL changed their state suddenly and randomly while performing the test.

- register log: This function gets information from the I2C slave (EPS) every 30 seconds for 45 minutes, prints it on the screen and creates a log file with the input name. You can select between NoPoL, Survival or Nominal mission configuration. In the Survival and Nominal mode, the PoL 2 is enabled for 9 min (20% magnetorquers' duty cycle)

The main opens the I2C bus and tries to find the I2C slave. If it has been successful, the main waits for the user to put the get, set, log or register command. The program will finish when the exit command is entered in the terminal.

In the following figure, we can see how the data is displayed in the terminal.

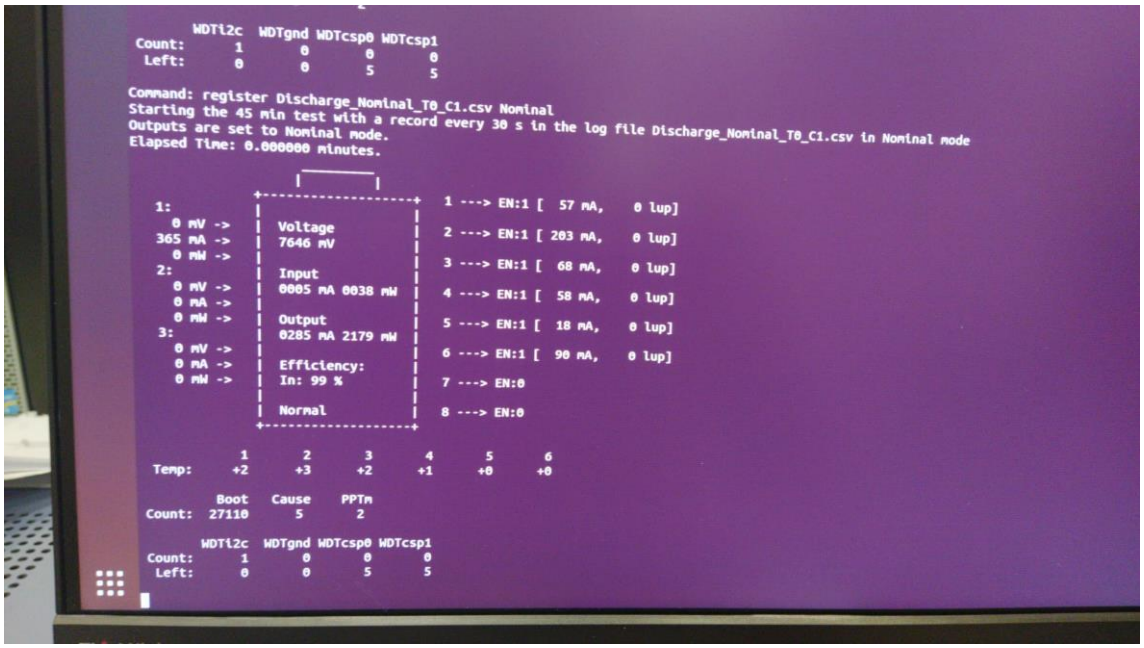



*Figure 4.10 - Example of data given by the EPS*

## 4.3. ENVIRONMENTAL TEST PREPARATION

Once checked the functionality of the code with our actual system and checked that the EPS answers the required data accordingly to our expectations, the environmental test can be carried out.

For this test, we needed a Thermal Vacuum Chamber (TVAC). During the course of the test, we subjected the EPS to different temperature and pressure stresses. It was an acceptance test to verify the operational charge and discharge temperature and the mission temperatures, as explained afterwards. Also, to characterize the voltage dependency on temperature.

Another important aspect of this test is that the EPS performed two mission modes during each discharge plateau. The time duration of the mission modes are 45 minutes, which are approximately one half of the orbit. This way, we could simulate a first approach of the EPS performance in a more realistic environment.

### 4.3.1. Pre-test work

Before the test was carried out, we designed a Perfboard (also called DOT PCB) with through hole resistors. The resistors were calculated regarding the power consumption and the voltage input.

$$W = V * I \rightarrow I = W/V \tag{4.1}$$

$$V = R * I \rightarrow R = \frac{V}{I} \rightarrow R = \frac{V^2}{W} \tag{4.2}$$

| Subsystems | PoL | Pin | Ground | Average Power (W) | Current (A) | Computed Resistors (ohms) | Actual Resistors (ohms) |
|---|---|---|---|---|---|---|---|
| OBC Mean Current(3.3V) | 4 | H1-48 | H2-29 | 0,17 | 0,052 | 64 | 62 |
| UHF/AIS and Helix Mean Current (3.3V) | 5 | H1-50 | H2-29 | 0,0532 | 0,016 | 205 | 200 |
| ADCS Mean Current (3.3V) | 6 | H1-52 | H2-29 | 0,3225 | 0,091 | 36 | 38 |
| COMMS Mean Current(5V) | 1 | H1-47 | H2-29 | 0,256 | 0,051 | 98 | 100 |
| MTQ Mean Current (5V) | 2 | H1-51 | H2-29 | 0,99* | 0,198* | 76 | 75 |
| Payload Mean Current(5V) | 3 | H1-49 | H2-29 | 0,3225 | 0,065 | 77 | 75 |
| TOTAL SUM | | | | | 0,473 | | |

*This current is the amperage of the three resistors added. Also, this is the only PoL that is enabled only 20% of the time (regarding its duty cycle), all the other resistors have been calculated including the duty cycle. The average power is also the sum of the three magnetorquers power consumption ignoring the 20% duty cycle.

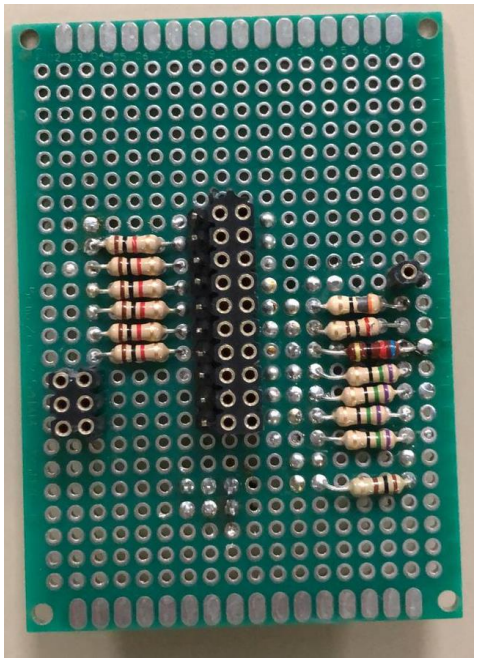


*Figure 4.11 - Perfboard with resistors..*

Moreover, before the test was carried out, we had to characterize the behavior of the EPS in charge and discharge modes. That was necessary because the EPS had problems in the charging interface, both in the photovoltaic input and the power supply input.

You can find attached in the Annex (Section 8.4) the EPS Characterization document, but as a conclusion, we can say that this EPS only has one configuration in which the batteries can charge.

We noticed that the EPS could probably have loading effect issues, because when we connected the Raspberry while charging, the power supply limited our current, indicating that it was necessary more current that the actual limitation, but it did not happen when it was charging without the Raspberry. This happened when the EPS was on while charging too.

So, the only way to charge the batteries regarding this is with the EPS off and Raspberry disconnected. We saw that the temperatures in normal conditions (20 ºC and 1 atmosphere) stabilize within 5 ºC hotter than the starting temperature. This is not applicable to the P8 charging test when there are loading effects but is not a problem when both the EPS and Raspberry are off.

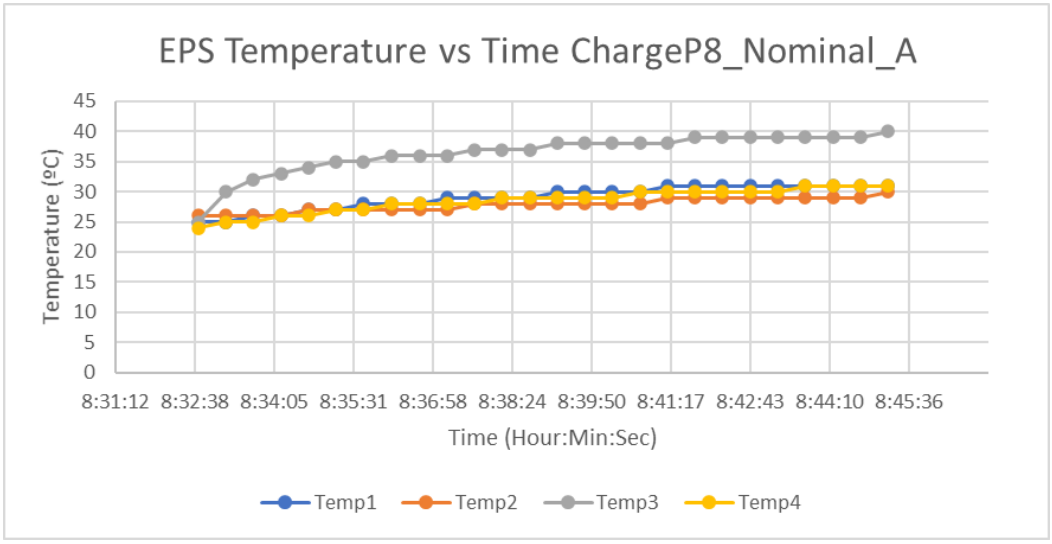



*Figure 4.12 - EPS Temperatures with EPS and Rasp connected*

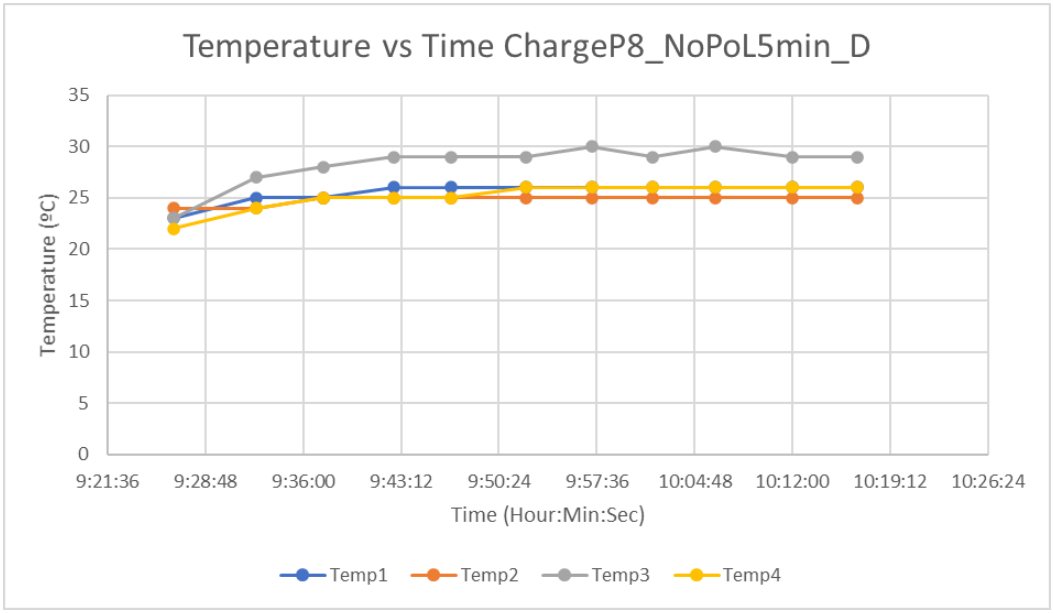


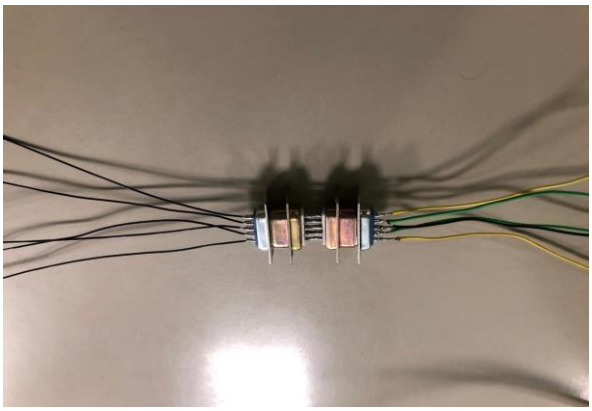

*Figure 4.13 - EPS Temperatures with EPS and Rasp disconnected*

As we can see, the charging setup that actually can charge the batteries remains inside the 5 ºC range, so we considered that we could carry on with the TVAC test.

### 4.3.2. TVAC Setup

After designing the Perfboard and once the characterization test was over, the next step was the actual preparation for the TVAC test.

The first thing we did was preparing the TVAC communication interface, because the EPS must communicate with the OBC, which was outside the TVAC.



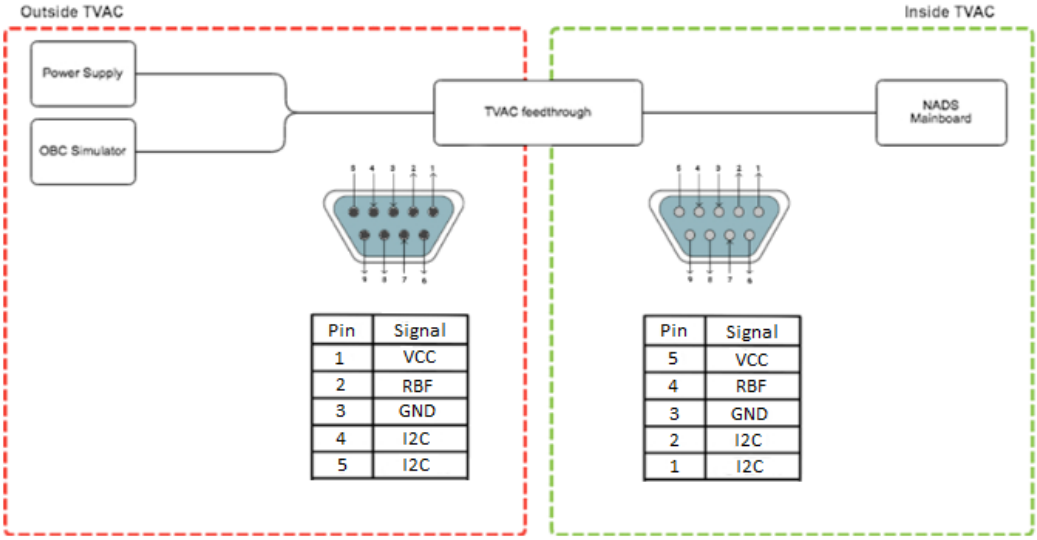



*Figure 4.14 - TVAC interface simulation and pinout specification*

The figure shows a simulation of how the interconnection between the raspberry and the EPS through the TVAC interface would be and also shows the assigned task for each pin. The VCC is the wire that connects the P8 charge pin of the EPS to the $V^+$ of the power supply.

The Remove Before Flight (RBF) allows to disconnect the EPS when shorted to ground, and turn it on when disconnecting the RBF from GND.

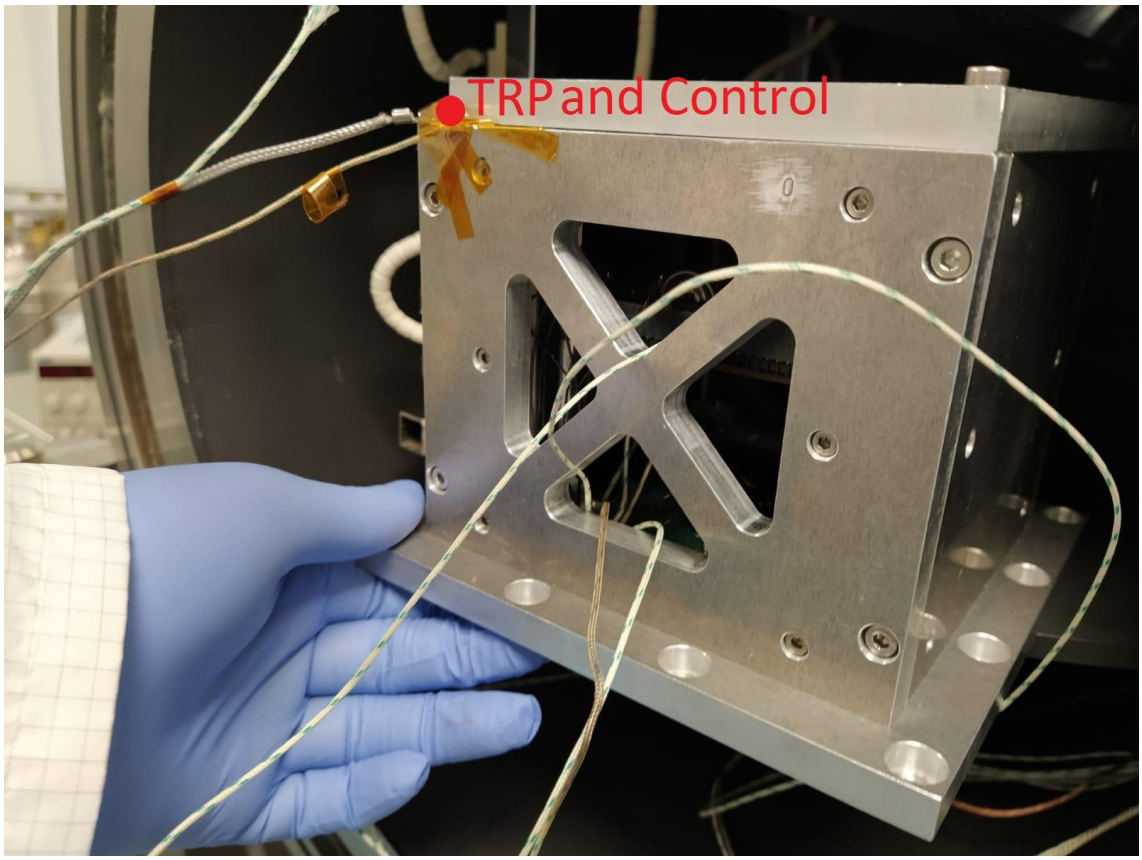
The ground wire is used for the charging when connected to the $V^-$ of the power supply. It is also used for the interconnection with the RBF as mentioned before and, additionally, must be connected to the Raspberry's GND for not having problems with the I2C communication.

Finally, the other two pins are the SDA and SCL I2C buses, that must be connected as specified in the EPS Wiring Communication (Section 4.2.1).

Subsequently, we entered in the NanoSatLab's cleanroom. A cleanroom is a controlled environment where pollutants like dust, airborne microbes, and aerosol particles are filtered out in order to provide the cleanest area possible.

There, we ensembled the EPS in the CubeSat structure and placed the thermocouples. The thermocouple is a sensor used to measure temperature. These sensors do not need to be powered because they generate a voltage (due to the thermoelectric effect) which is dependent on the temperature, so there is no probability of short-circuit in the EPS board.

Afterwards, we connected the resistors of the Perfboard to their corresponding PoL and encapsulated all in the aluminum box shown below.



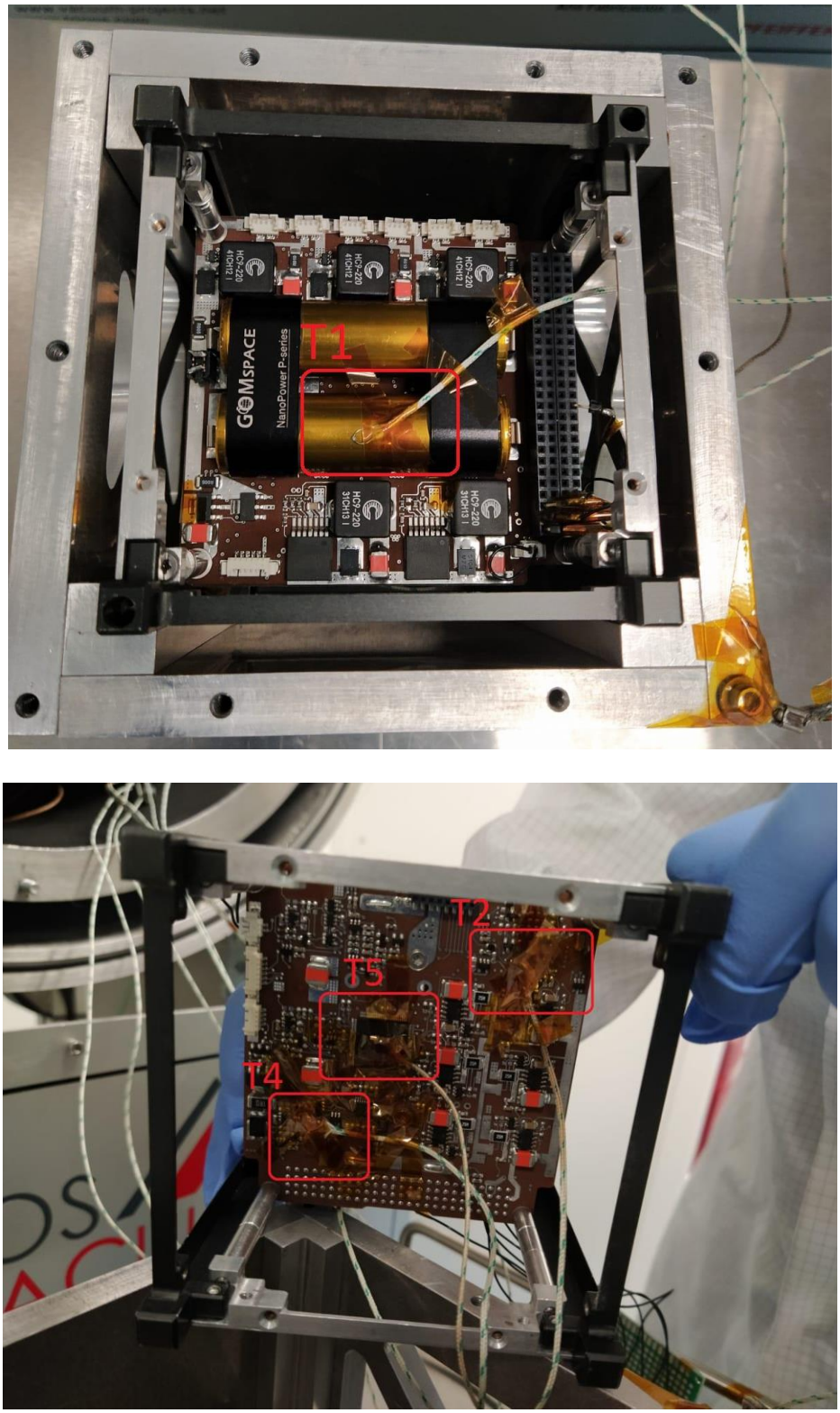



*Figure 4.15 - Thermocouples placing*

Thermocouples T1, T2, and T4 where were used to double-check the EPS sensors while T5 was controlling the temperature of the processor of the EPS. The Temperature Reference Point is the thermocouple that will actually control

the TVAC heating and cooling system, so as it is the most important thermocouple, the Control thermocouple is placed in the same position. If the temperatures are quite different between these two sensors, the test must be stopped. Notice that all the thermocouples are fastened with Kapton, as it is highly resistant to extreme temperatures.

The TRP is placed in the aluminum box because it is safer this way. If the controller of the heating and cooling system does not work properly or the TRP starts getting wrong measurements, the inner lamps of the TVAC could overheat the batteries in a few seconds and could be extremely dangerous.

As all the heating process is made using thermal radiation, if the lamps illuminate the aluminum box, it will increase in temperature and radiate to the EPS, so the process will be much slower and hence, safer.

Eventually, we closed the TVAC and set the profile it had to follow. The profile, as explained in the Verification Tests (Section 1.3), has two cycles regarding the operational temperatures acceptance test and two cycles regarding the mission temperatures acceptance test. The test temperatures for the operational temperatures acceptance test are 5 to 35 ºC for the charge test and 0 to 50 ºC for the discharge one, whilst the test temperatures for the mission temperatures acceptance test are equal in the charge test, but 0 to 45ºC for the discharge one.

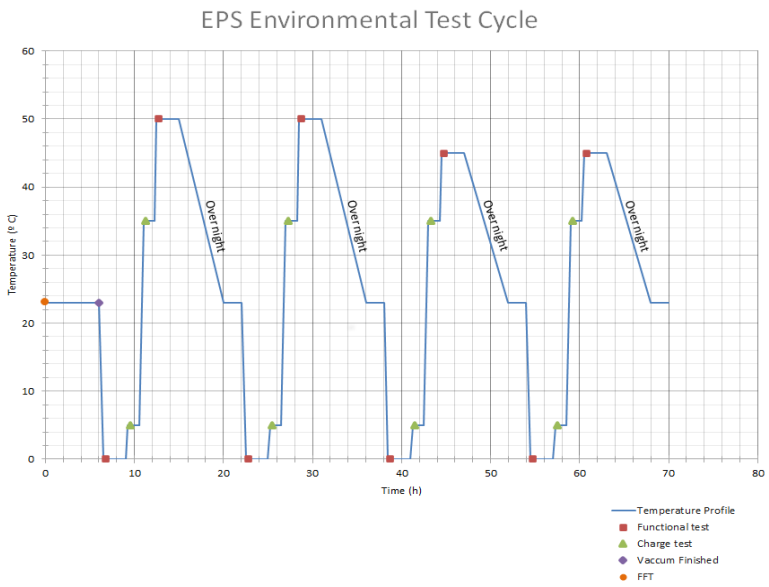



*Figure 4.16 - TVAC test profile*

All the information summarized in this section can be found with a more in detail explanation in the Annex (Section 8.5) where the test specifications and procedures are disclosed.

# 5. RESULTS

The Results section will evaluate the data gathered during the TVAC test campaign, but as said before, it will not describe all the information obtained from the test. This section aims to show the most important gathered data and which is the meaning of these results. If you are willing to read a more in detail document about the data extracted from the test, you can refer to the Annex (Section 8.6).

The first important information extracted is that, as there is no thermal convection between the air and the EPS, because it is in vacuum, the EPS reaches a temperature much higher than it did when it was at standard conditions.

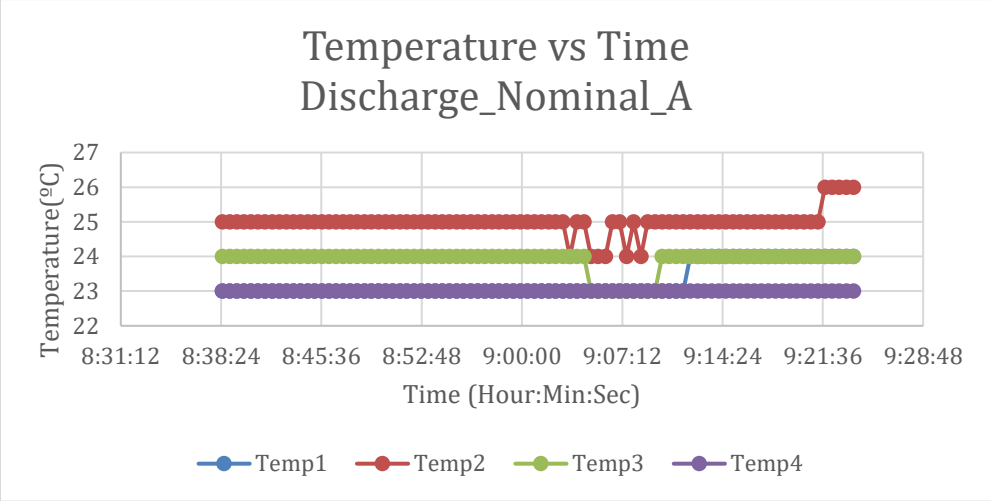



*Figure 5.1 - Temperature plot at 1 atm*

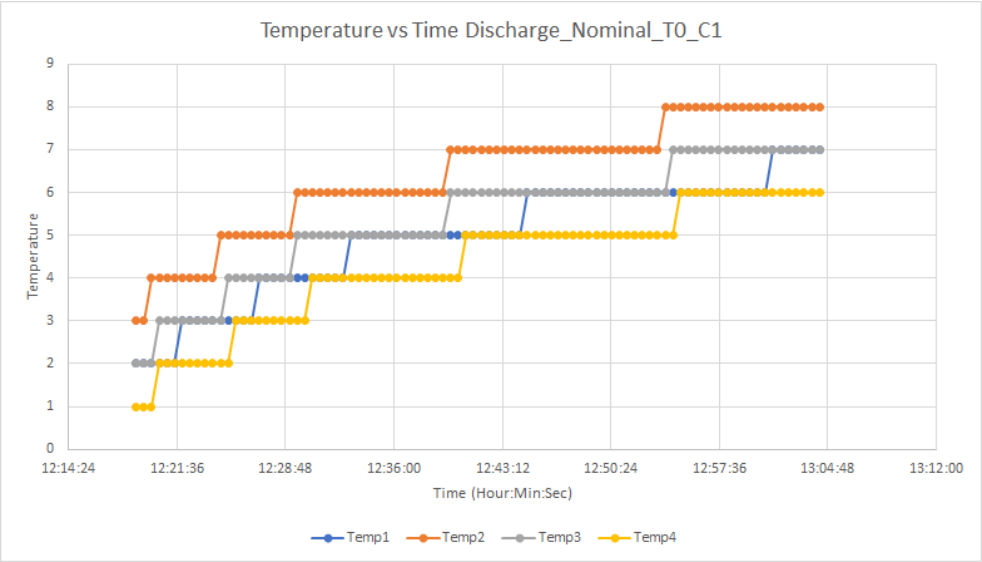



*Figure 5.2 - Temperature plot at $10^{-5}$ mbar*

Even though the temperatures are not the same, we can observe that the 45-minute nominal test at 1 atm hardly increases the temperature whilst the one at low pressure increases it as much as 5ºC. In addition to this new handicap, there is the fact that the temperature can only decrease thanks to the radiation properties of the EPS, so the temperature will decrease much slower than it made at standard pressure. This means that the subsystem must be operated cautiously when switched on during large periods of time. It also has to be said that operating the EPS in the hot condition is dangerous because the operational temperature of the batteries could be exceeded.

The next important behavior seen is the fact that when the system disconnects the magnetorquers PoL, as its duty cycle is 20% (9 minutes from the 45 minute test), there is a hop in voltage, which suddenly increases.

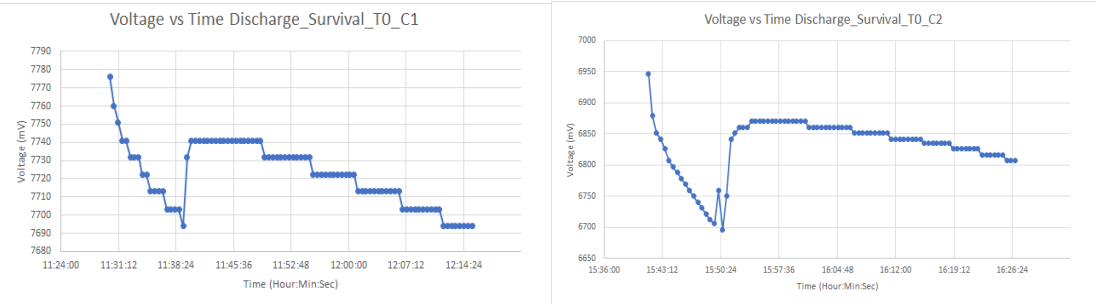



*Figure 5.3 - Comparison between different voltage hops*

This change in voltage is believed to be related to a charge effect of the internal resistance of the battery. The next figure aims to show the hypothetical electric scheme that matches the results shown in Figure 5.3.

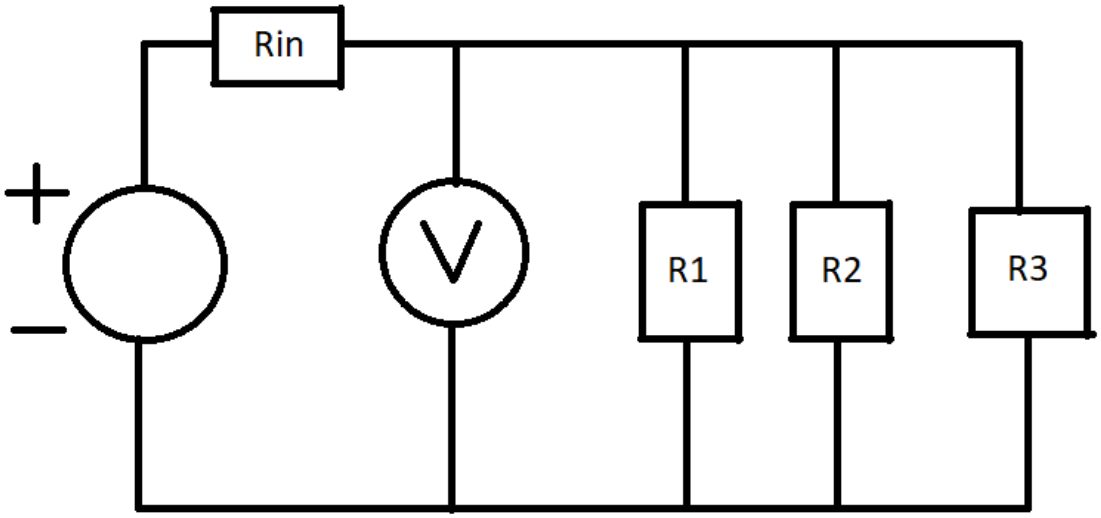



*Figure 5.4 - Electrical Scheme of the loaded EPS*

The resistor $R_{in}$ is the one that represents the internal resistance of the EPS and the R# are the loads powered by the batteries.

One of these resistors we could imagine that are the magnetorquers, and we know that at some point, they will disconnect, making the overall resistance of the loads higher (as they are in parallel). This fact will increase the voltage drop in the load resistors, and thus, the voltage read by the voltmeter.

Moreover, as seen in Figure 5.3, the hops are different in magnitude even for the same mode and temperature, which is expected as this hop depends on the batteries voltage. As the voltage of the batteries increases, the hop will increase in magnitude also, because the loading effect is proportional to this voltage. This is not the behavior seen, as the hop in the lower voltage is higher, so this leads to think that the internal resistance varies with respect to the current voltage stored in the batteries.

The next information extracted from the gathered data is that the rate of discharge in cold and hot temperatures are quite different. The behavior is like a parable in the cold condition, whilst in the hot condition is quite linear.

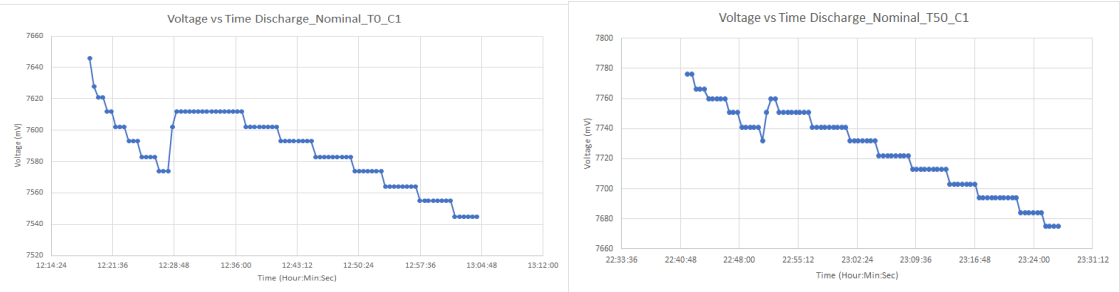



*Figure 5.5 -Comparison between the hot and cold conditions voltage vs time plots*

In order to understand this, you must know that usually the resistance of the resistors grows with respect to the temperature increase. The plots behavior can be explained because the temperature is low in the discharge mode at 0 ºC and the thermal radiation properties are poorer than the 50 ºC discharge mode. The temperature rises very quickly at the beginning, making thermal conduction between the resistor and the PCB the most important heat exchange factor. As the resistor heats the PCB, the thermal conduction properties decrease, but the radiation properties increase as the temperature gets higher, until it reaches thermal equilibrium and the rate of discharge stabilizes.

For the hot case, the temperature is currently high, so the radiation properties are better than in the cold case. Also, as the temperature is higher, the thermal equilibrium is closer than in the cold case. That is why the discharge rate is more stable than in the cold condition and the voltage vs time behaves linearly.

The last information extracted is the difference in the rate of charge.
As there was no clear pattern in the behavior of this property, a characterization of the charge rate has been made for the extraction of conclusions.

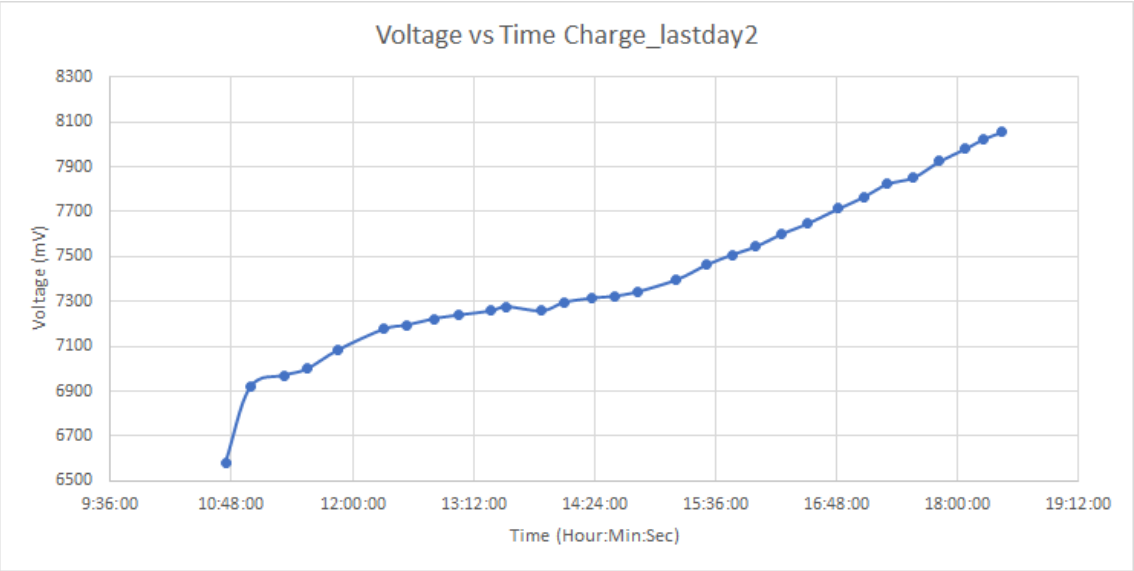



*Figure 5.6 - Charge Characterization*

As can be seen in the plot, the charge rate is not constant during all the voltage frames. At the beginning the charge rate is huge, reaching 1340 mV/h in the 6.5-6.9 V frame. The worst charge rate is the one comprised between the 6.9-7.3 V fame, which is about 100 mV/h. Finally, in the last stage, from 7.4 to 8.1 V, the charge rate is about 200 mV/h.

# 6. CONCLUSIONS

The project conclusions must be divided in two parts, as well as the report objectives described in the Document Goals (Section 1.4). The conclusions will check if the proposed objectives have succeeded.

The first part corresponds to the functional test, which is testing if the EPS works fine or at least realizes the functions we are interested in. These functions are basically: return all the housekeeping (voltage, temperatures, PoLs, etc.) and enabling/disabling the different PoLs. In addition, the EPS must be capable of charging its batteries while it is on and perform discharge tests during charge period.

Even though some I2C protocol functionalities are not well implemented in this GomSpace product (for example, the master cannot obtain the slave address and you have to debug the EPS with minicom to obtain it), if the structure given in the datasheet is followed accurately, all the required functionalities work well.

The received data is correct, and the PoLs can be enabled/disabled almost instantaneously. However, there is a bad functionality in the PoLs because every single PoL can suddenly change its state from enabled to disabled or vice versa.

Moreover, the charging interface only works with the EPS off and without the OBC (in this case the Raspberry) connected to ground. If this is not accomplished, it can be seen a current leakage and the power supply will short-circuit showing that not all the current is delivered to the battery, and then, it will fail to charge.

The only way to charge it is to disconnect all devices, including the EPS itself. Due to the aforementioned behavior, I must say that the functional test is not successful. Regardless of that, the environmental test has been carried out in order to study the EPS behavior under different temperatures.

The second part corresponds to the environmental test, which consist in testing the functionality of the EPS under different environmental conditions and observe the voltage dependency on temperature.

As explained before, all the test setup and procedure is explained in the Annex (Section 8.5) in the TSTP document, and all the acquired data is shown in the Annex (Section 8.6) in the TRPT document. The summarized versions can be found in the Section 4.3 and 5 respectively.

There are four important issues discovered from the environmental test. The first thing is that there is a charge effect in the batteries which causes hops in voltage.

This can be important when heavily loading the EPS PoL, because the huge hop could make the EPS to enter in critical mode, which would switch off all PoLs. Also, you can think that the EPS voltage is enough to do its duties, but when loading it the voltage would decrease and lead to problems.

The second fact is the temperature, which in the 1 atm condition practically does not increase but, in the vacuum condition, it increases importantly due to the absence of thermal convection between the air and the EPS.

The third one is the change of rate of discharge between the hot and cold temperature, which, as said before, is due to the difference in radiation and conduction properties and the difference in proximity to the thermal equilibrium temperature.

The fourth fact is that the charge rate is not constant with respect to the current voltage of the batteries. Depending on the current voltage of the batteries, they will charge faster or slower.

As a conclusion, I could say that the environmental test has been successful as all the functionalities that were working at standard temperature, they also did in the cold and hot temperature. In addition, voltage and temperature data has been extracted successfully. Nevertheless, the charging interface does not work while the EPS is on or the OBC is connected. Also, the system reached temperatures close to the operational ones, which is not acceptable. This means this test has not been successful neither.

As an overall conclusion I must say that the EPS has not successfully passed the tests. All these failures could be explained by the time degradation of the batteries and components of the EPS. This subsystem was not a new one and has been used before, so there is a chance that the problems could solve changing to a new device.

However, the information gathered will be extremely useful for future work. All the procedure that has been developed during the course of this project had never been done before in the NanoSatLab for this specific type of device, so all this procedure can be reused with the EPS Proto-Flight Model (PFM).

The EPS PFM test has already been scheduled and the data collected in this project will be especially useful for comparing the results that will be extracted from the new test. Moreover, the results and conclusions obtained in this project may also help other researchers facing similar problems.

# 7. REFERENCE DOCUMENTS

[1] Basis on I2C communications. Retrieved on 02/04/2020. https://learn.sparkfun.com/tutorials/i2c/all
[2] GomSpace 2013, NanoPower P-series Datasheet, P31u V8.0, GomSpace, Aalborg, Denmark
[3] History of CubeSats. Retrieved on 07/05/2020. https://www.bbc.com/mundo/noticias-49031317
[4] History of CubeSats (2). Retrieved on 07/05/2020. https://en.wikipedia.org/wiki/CubeSat
[5] State of the Art and CubeSats Specifications. Retrieved on 07/05/2020. http://www.ece3sat.com/cubesatmodules/eps/
[6]EPS Architecture. Retrieved on 19/08/2020. https://digitalcommons.usu.edu/cgi/viewcontent.cgi?article=1136&context=smallsat
[6]Li-Ion vs Li-Po. Retrieved on 21/08/2020. https://blog.ravpower.com/2017/06/lithium-ion-vs-lithium-polymer-batteries/
[7] Camps, Adriano. Muñoz, Joan Francesc. Ruiz de Azúa, Joan Adrià. 29/11/2017, 3C4_MDD_20170917_v1.2, NanoSatLab, Barcelona, Spain
[8] Camps, Adriano. Ruiz de Azúa, Joan Adrià. 17/06/2018, 3C4_DDF_20170917_v1.8, NanoSatLab, Barcelona, Spain

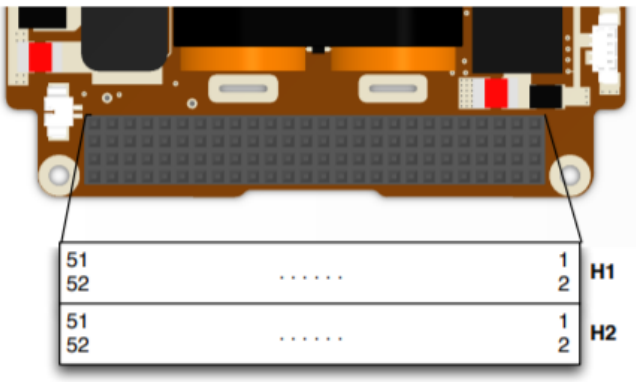
# 8. ANNEX

## 8.1. EPS Pinout

EPS Pinout extracted from EPS datasheet [2]

**Stack Connector**

| Pin# | Mnemonic | Dir | Description |
|---|---|---|---|
| H1-32 | 5V_in | I | 5V battery charge input (Same as P8 pin 6) |
| H1-41 | I2C-SDA | I/O | I2C serial data |
| H1-43 | I2C-SCL | I/O | I2C serial clock |
| H1-47 | OUT-1 | O | latch-up protected output |
| H1-49 | OUT-2 | O | latch-up protected output |
| H1-51 | OUT-3 | O | latch-up protected output |
| H1-48 | OUT-4 | O | latch-up protected output |
| H1-50 | OUT-5 | O | latch-up protected output |
| H1-52 | OUT-6 | O | latch-up protected output |
| H2-25 | +5V | O | Permanent 5V output |
| H2-26 | +5V | O | Permanent 5V output |
| H2-27 | +3.3V | O | Permanent 3.3V output |
| H2-28 | +3.3V | O | Permanent 3.3V output |
| H2-29 | GND | O | Power ground |
| H2-30 | GND | O | Power ground |
| H2-31 | AGND | O | Analogue ground |
| H2-32 | GND | O | Power ground |
| H2-45 | V_BAT | O | Battery voltage |
| H2-46 | V_BAT | O | Battery voltage |




*Stack connector pinout*

## 8.2. EPS I2C Ports (Commands) example

| Mnemonic | Port | Request/Reply Data | Bytes | Description |
|---|---|---|---|---|
| GET_HK_1 (backwards compatible) | 8 | empty | 0 | Send empty packet to request backwards compatible housekeeping struct. |
| | | struct hkparam_t; | struct | Reply: Data-structure (see below) |
| GET_HK_2 (p31u-8 format) | 8 | uint8_t type = 0 | 1 | Send packet of length = 1 with type = 0 to request p31u-8 housekeeping struct. |
| | | struct eps_hk_t; | struct | Reply: Data-structure (see below) |
| GET_HK_2_VI | 8 | uint8_t type = 1 | 1 | Send packet of length = 1 with type = 1 to request voltage and current subset of HK_2 |
| | | struct eps_hk_vi_t; | struct | Reply: Data-structure (see below) |
| GET_HK_2_OUT | 8 | uint8_t type = 2 | 1 | Send packet of length = 1 with type = 2 to request output switch data subset of HK_2 |
| | | struct eps_hk_out_t; | struct | Reply: Data-structure (see below) |
| GET_HK_2_WDT | 8 | uint8_t type = 3 | 1 | Send packet of length = 1 with type = 3 to request wdt data subset of HK_2 |
| | | struct eps_hk_wdt_t; | struct | Reply: Data-structure (see below) |
| GET_HK_2_BASIC | 8 | uint8_t type = 4 | 1 | Send packet of length = 1 with type = 4 to request the basic data subset of HK_2 |
| | | struct eps_hk_basic_t; | struct | Reply: Data-structure (see below) |
| SET_OUTPUT | 9 | uint8 output_byte; | 1 | Set output switch states by a bitmask where "1" means the channel is switched on and "0" means it is switched off. LSB is channel 1, next bit is channel 2 etc. (Quadbat switch and heater cannot be controlled through this command) [NC NC 3.3V3 3.3V2 3.3V1 5V3 5V2 5V1] |
| | | none | x | No reply to this command |
| SET_SINGLE_OUTPUT | 10 | uint8 channel, uint8 value; int16 delay | 4 | Set output %channel% to value %value% with delay %delay%, Channel (0-5), Quadbat heater (6), Quadbat switch (7) Value 0 = Off, 1 = On Delay in seconds. |
| | | none | x | No reply to this command |
| SET_PV_VOLT | 11 | uint16 voltage1, uint16 voltage2, uint16 voltage3; | 6 | Set the voltage on the photo-voltaic inputs V1, V2, V3 in mV. Takes effect when MODE = 2, See SET_PV_AUTO. Transmit voltage1 first and voltage3 last. |
| | | none | x | No reply to this command |
| SET_PV_AUTO | 12 | uint8 mode; | 1 | Sets the solar cell power tracking mode: MODE = 0: Hardware default power point MODE = 1: Maximum power point tracking MODE = 2: Fixed software powerpoint, value set with SET_PV_VOLT, default 4V |
| | | none | x | No reply to this command |
| SET_HEATER_AUTO | 13 | uint8 mode; | 1 | Sets the heater auto mode on or off: MODE = 0: Heater auto mode off MODE = 1: Heater auto mode on MODE = any other value: Heater auto mode unchanged. Auto mode will switch on heater when battery temperature is -5 degC and off when battery temperature is +5 degC. |
| | | uint8 mode; | 1 | Command replies with heater auto mode. To query , simple send any other value that 0 or 1. |

## 8.3. C Script

```
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <unistd.h>                    //Needed for I2C port and
for sleep function
#include <fcntl.h>                     //Needed for I2C port
#include <sys/ioctl.h>                 //Needed for I2C port
#include <linux/i2c-dev.h>       //Needed for I2C port
#include <arpa/inet.h>
#include <math.h>

/**
 * Union for storing the block of telemetry values coming from
the EPS. HK version 2.
 */
typedef union __attribute__ ((__packed__)) eps_hk_t
{
      unsigned char raw[133]; /**< Unformatted GOM EPS telemetry
*/
      struct __attribute__((packed))
      {
            unsigned short commandReply; /**< reply of the last
command */
            unsigned short vboost[3]; /**< Voltage of boost
converters [mV] [PV1, PV2, PV3] */
            unsigned short vbatt; /**< Voltage of battery [mV] */
            unsigned short curin[3]; /**< Current in [mA] */
            unsigned short cursun; /**< Current from boost
converters */
            unsigned short cursys; /**< Current out of battery */
            unsigned short reserved1; /**< Reserved for future
use */
            unsigned short curout[6]; /**< Current out [mA] */
            unsigned char output[8]; /**< Status of outputs */
            unsigned short output_on_delta[8]; /**< Time till
power on */
            unsigned short output_off_delta[8]; /**< Time till
power off */
            unsigned short latchup[6]; /**< Number of latch-ups
*/
            unsigned int wdt_i2c_time_left; /**< Time left on I2C
wdt */
            unsigned int wdt_gnd_time_left; /**< Time left on I2C
wdt */
            unsigned char wdt_csp_pings_left[2]; /**< Pings left
on CSP wdt */
            unsigned int counter_wdt_i2c; /**< Number of WDT I2C
reboots */
            unsigned int counter_wdt_gnd; /**< Number of WDT GND
reboots */
            unsigned int counter_wdt_csp[2]; /**< Number of WDT
CSP reboots */
            unsigned int counter_boot; /**< Number of EPS reboots
*/
```

```
          short temp[6]; /**< Temperature sensors [0 = TEMP1,
TEMP2, TEMP3, TEMP4, BATT0, BATT1] */
          unsigned char bootcause; /**< Cause of last EPS reset
*/
          unsigned char battmode; /**< Mode for battery [0 =
normal, 1 = undervoltage, 2 = overvoltage] */
          unsigned char pptmode; /**< Mode of PPT tracker */
          unsigned short reserved2;
     } fields; /**< Struct with individual fields of GOM EPS
telemetry. HK version 2. 92 */
} eps_hk_t;

void eps_hk_print(eps_hk_t * hk) {
     unsigned int p_in_1 = (float) hk->fields.vboost[0] *
(float) hk->fields.curin[0] / 1000.0;
     unsigned int p_in_2 = (float) hk->fields.vboost[1] *
(float) hk->fields.curin[1] / 1000.0;
     unsigned int p_in_3 = (float) hk->fields.vboost[2] *
(float) hk->fields.curin[2] / 1000.0;
     unsigned int p_sun = (float) hk->fields.cursun * (float)
hk->fields.vbatt / 1000.0;
     unsigned int p_sys = (float) hk->fields.cursys * (float)
hk->fields.vbatt / 1000.0;
     unsigned int eff_in = (100.0 * (float) p_sun) / ((float)
p_in_1 + (float) p_in_2 + (float) p_in_3);
     if (eff_in >= 100)
          eff_in = 99;

     printf("                      _________\r\n");
     printf("                    |         |\r\n");
     printf("               +--------------------+   1 --->
EN:%u [%4u mA, %4u lup]\r\n", hk->fields.output[0], hk-
>fields.curout[0], hk->fields.latchup[0]);
     printf("   1:          |                    |\r\n");
     printf("  %4u mV ->    |  Voltage           |   2 ---> EN:%u
[%4u mA, %4u lup]\r\n", hk->fields.vboost[0], hk-
>fields.output[1], hk->fields.curout[1], hk->fields.latchup[1]);
     printf("  %4u mA ->    |  %04u mV           |\r\n", hk-
>fields.curin[0], hk->fields.vbatt);
     printf("  %4u mW ->    |                    |   3 ---> EN:%u
[%4u mA, %4u lup]\r\n", p_in_1, hk->fields.output[2], hk-
>fields.curout[2], hk->fields.latchup[2]);
     printf("   2:          |  Input             |\r\n");
     printf("  %4u mV ->    |  %04u mA %04u mW   |   4 ---> EN:%u
[%4u mA, %4u lup]\r\n", hk->fields.vboost[1], hk->fields.cursun,
p_sun, hk->fields.output[3], hk->fields.curout[3], hk-
>fields.latchup[3]);
     printf("  %4u mA ->    |                    |\r\n", hk-
>fields.curin[1]);
     printf("  %4u mW ->    |  Output            |   5 ---> EN:%u
[%4u mA, %4u lup]\r\n", p_in_2, hk->fields.output[4], hk-
>fields.curout[4], hk->fields.latchup[4]);
     printf("   3:          |  %04u mA %04u mW   |\r\n", hk-
>fields.cursys, p_sys);
```

```
	printf("  %4u mV ->   |                   |   6 ---> EN:%u
[%4u mA, %4u lup]\r\n", hk->fields.vboost[2], hk-
>fields.output[5], hk->fields.curout[5], hk->fields.latchup[5]);
	printf("  %4u mA ->   |  Efficiency:      |\r\n", hk-
>fields.curin[2]);
	printf("  %4u mW ->   |  In: %2u %%         |   7 --->
EN:%u\r\n", p_in_3, eff_in, hk->fields.output[6]);
	printf("              |                   |\r\n");
	printf("              |  ");
	switch(hk->fields.battmode) {
	case 1: printf("CRITICAL"); break;
	case 2: printf("SAFEMODE"); break;
	case 3: printf("Normal  "); break;
	case 4: printf("Full    "); break;
	}
	printf("         |   8 ---> EN:%u\r\n", hk-
>fields.output[7]);
	printf("              +-------------------+\r\n");
	printf("\r\n");
	printf("             1       2       3       4       5
6\r\n");
	printf("  Temp:   %+4d    %+4d    %+4d    %+4d    %+4d
%+4d\r\n", hk->fields.temp[0], hk->fields.temp[1], hk-
>fields.temp[2], hk->fields.temp[3], hk->fields.temp[4], hk-
>fields.temp[5]);
	printf("\r\n");
	printf("          Boot   Cause    PPTm\r\n");
	printf(" Count:  %5u  %5x   %5u\r\n", hk-
>fields.counter_boot, hk->fields.bootcause, hk->fields.pptmode);
	printf("\r\n");
	printf("        WDTi2c  WDTgnd WDTcsp0 WDTcsp1\r\n");
	printf(" Count:  %5u   %5u   %5u   %5u\r\n", hk-
>fields.counter_wdt_i2c, hk->fields.counter_wdt_gnd, hk-
>fields.counter_wdt_csp[0], hk->fields.counter_wdt_csp[1]);
	printf("  Left:  %5u   %5u   %5u   %5u\r\n", hk-
>fields.wdt_i2c_time_left, hk->fields.wdt_gnd_time_left, hk-
>fields.wdt_csp_pings_left[0], hk-
>fields.wdt_csp_pings_left[1]);
	printf("\r\n");
}

void eps_hk_nth(eps_hk_t *hk){
	int i;

	/* arrays with 3 */
	for (i = 0; i < 3; i++) {
		hk->fields.vboost[i] = ntohs(hk->fields.vboost[i]);
		hk->fields.curin[i] = ntohs(hk->fields.curin[i]);
	}

	/* arrays with 6 */
	for (i = 0; i < 6; i++) {
		hk->fields.curout[i] = ntohs(hk->fields.curout[i]);
		hk->fields.temp[i] = ntohs(hk->fields.temp[i]);
		hk->fields.latchup[i] = ntohs(hk->fields.latchup[i]);
	}
```

```
      /* arrays with 8 */
      for (i = 0; i < 8; i++) {
            hk->fields.output[i] = hk->fields.output[i];
            hk->fields.output_on_delta[i] = ntohl(hk-
>fields.output_on_delta[i]);
            hk->fields.output_off_delta[i] = ntohl(hk-
>fields.output_off_delta[i]);
      }

      hk->fields.vbatt = ntohs(hk->fields.vbatt);
      hk->fields.cursun = ntohs(hk->fields.cursun);
      hk->fields.cursys = ntohs(hk->fields.cursys);
      hk->fields.wdt_i2c_time_left = ntohl(hk-
>fields.wdt_i2c_time_left);
      hk->fields.wdt_gnd_time_left = ntohl(hk-
>fields.wdt_gnd_time_left);
      hk->fields.counter_boot = ntohl(hk->fields.counter_boot);
      hk->fields.counter_wdt_i2c = ntohl(hk-
>fields.counter_wdt_i2c);
      hk->fields.counter_wdt_gnd = ntohl(hk-
>fields.counter_wdt_gnd);
      hk->fields.counter_wdt_csp[0] = ntohl(hk-
>fields.counter_wdt_csp[0]);
      hk->fields.counter_wdt_csp[1] = ntohl(hk-
>fields.counter_wdt_csp[1]);
}


// GomGetTelemetry
int GomGetTelemetry(int i2c_handler, eps_hk_t *hk)
{
      unsigned char buffer[60];
      //----- WRITE BYTES -----
      buffer[0] = 0x08;
      buffer[1] = 0x00;
      int length = 2;             //<<< Number of bytes to write
      if (write(i2c_handler, buffer, length) != length)
      //write() returns the number of bytes actually written, if
it doesn't match then an error occurred (e.g. no response from
the device)
      {
            /* ERROR HANDLING: i2c transaction failed */
            printf("Failed to write to the i2c bus.\n");
      }
      usleep(5000);
      //----- READ BYTES -----              //<<< Number of
bytes to read
      //read() returns the number of bytes actually read, if it
doesn't match then an error occurred (e.g. no response from the
device)
      if (read(i2c_handler, hk->raw , sizeof(eps_hk_t)) !=
sizeof(eps_hk_t)) {
            //ERROR HANDLING: i2c transaction failed
            printf("Failed to read from the i2c bus.\n");
            return -1;
```

```
      } else {
            eps_hk_nth(hk);
            return 0;
      }

}

void SetOutputStatus(int i2c_handler, int output, u_int8_t
status,eps_hk_t * hk){

      unsigned char buffer[2];
      //Get and Modify Housekeeping in order to set output byte
      GomGetTelemetry(i2c_handler,hk);
      eps_hk_nth(hk);
      usleep(5000);
      //Set the given output to the given status
      hk->fields.output[output-1]=status;
      printf("Output %d is set to %d\n", output, status);
      //----- WRITE BYTES -----
      buffer[0] = 0x09;
      buffer[1]=pow(2,5)*hk->fields.output[5]+pow(2,4)*hk-
>fields.output[4]+pow(2,3)*hk->fields.output[3]+pow(2,2)*hk-
>fields.output[2]+pow(2,1)*hk->fields.output[1]+pow(2,0)*hk-
>fields.output[0];
      int length = 2;            //<<< Number of bytes to write
      if (write(i2c_handler, buffer, length) != length)
      //write() returns the number of bytes actually written, if
it doesn't match then an error occurred (e.g. no response from
the device)
      {
            /* ERROR HANDLING: i2c transaction failed */
            printf("Failed to write to the i2c bus.\n");
      }
      else {
            usleep(500000);
            if (GomGetTelemetry(i2c_handler, hk) == 0) {
                  eps_hk_print(hk);
            }
      }
}



void representTime(struct tm* ts, char* str)
{
      sprintf(str, "%d-%d-%d %d:%d:%d", ts->tm_mday, ts->tm_mon
+ 1, ts->tm_year + 1900, ts->tm_hour, ts->tm_min, ts->tm_sec);
}
int file_log(eps_hk_t * hk) {

      time_t t;
         struct tm* ts;
         char buffer[20];
      char filename[]="Charge_lastday2.csv";
         // creating file pointer to work with files
         FILE* fptr;
```

```
        FILE* fptrr;
        //looking if the file exists
        fptrr = fopen(filename, "r");
        if (fptrr == NULL)
        {
                // opening file in writting mode
                fptr = fopen(filename, "w");

                // exiting program
                if (fptr == NULL) {
                        printf("Error while creating log!");
                        return -1;
                }
                fprintf(fptr, "Time and
Date;Vbat(mV);Temp1;Temp2;Temp3;Temp4;OutputState1;OutputState2;
OutputState3;OutputState4;OutputState5;OutputState6;CurOut1(mA);
CurOut2(mA);CurOut3(mA);CurOut4(mA);CurOut5(mA);CurOut6(mA);Boot
Count\n");
                fclose(fptr);
        }
        else
        {
                fclose(fptrr);
        }
        // opening file in appending mode
        fptr = fopen(filename, "a");

        // exiting program
        if (fptr == NULL) {
                printf("Error, you must close the log file!");
                return -1;
        }

        t = time(0);
        ts = localtime(&t);
        representTime(ts, buffer);
        fprintf(fptr,
"%s;%04u;%+4d;%+4d;%+4d;%+4d;%u;%u;%u;%u;%u;%u;%4u;%4u;%4u;%4u;%
4u;%4u;%5u\n", buffer, hk->fields.vbatt,hk->fields.temp[0],hk-
>fields.temp[1],hk->fields.temp[2],hk->fields.temp[3],hk-
>fields.output[0],hk->fields.output[1],hk->fields.output[2],hk-
>fields.output[3],hk->fields.output[4],hk->fields.output[5],hk-
>fields.curout[0],hk->fields.curout[1],hk->fields.curout[2],hk-
>fields.curout[3],hk->fields.curout[4],hk->fields.curout[5],hk-
>fields.counter_boot);

        fclose(fptr);

      return 0;
}

int customed_file_log(eps_hk_t * hk, char file_name[30]) {

      time_t t;
      struct tm* ts;
      char buffer[20];
```

```
	// creating file pointer to work with files
	FILE* fptr;
	FILE* fptrr;
	//looking if the file exists
	fptrr = fopen(file_name, "r");
	if (fptrr == NULL)
	{
		// opening file in writting mode
		fptr = fopen(file_name, "w");

		// exiting program
		if (fptr == NULL) {
				printf("Error while creating log!");
				return -1;
		}
		fprintf(fptr, "Time and
Date;Vbat(mV);Temp1;Temp2;Temp3;Temp4;OutputState1;OutputState2;
OutputState3;OutputState4;OutputState5;OutputState6;CurOut1(mA);
CurOut2(mA);CurOut3(mA);CurOut4(mA);CurOut5(mA);CurOut6(mA);Boot
Count\n");
		fclose(fptr);
	}
	else
	{
		fclose(fptrr);
	}
	// opening file in appending mode
	fptr = fopen(file_name, "a");

	// exiting program
	if (fptr == NULL) {
		printf("Error, you must close the log file!");
		return -1;
	}

	t = time(0);
	ts = localtime(&t);
	representTime(ts, buffer);
	fprintf(fptr,
"%s;%04u;%+4d;%+4d;%+4d;%+4d;%u;%u;%u;%u;%u;%u;%4u;%4u;%4u;%4u;%
4u;%4u;%5u\n", buffer, hk->fields.vbatt,hk->fields.temp[0],hk-
>fields.temp[1],hk->fields.temp[2],hk->fields.temp[3],hk-
>fields.output[0],hk->fields.output[1],hk->fields.output[2],hk-
>fields.output[3],hk->fields.output[4],hk->fields.output[5],hk-
>fields.curout[0],hk->fields.curout[1],hk->fields.curout[2],hk-
>fields.curout[3],hk->fields.curout[4],hk->fields.curout[5],hk-
>fields.counter_boot);

	fclose(fptr);

	return 0;
}

void SetOutput(int i2c_handler, char mode[30], int i){

	unsigned char buffer[2];
```

```
      //Get and Modify Housekeeping in order to set output byte
      printf("Outputs are set to %s mode.\n",mode);
      //----- WRITE BYTES -----
      buffer[0] = 0x09;

      if(strcmp(mode,"NoPoL") == 0){


      buffer[1]=pow(2,5)*0+pow(2,4)*0+pow(2,3)*0+pow(2,2)*0+pow(
2,1)*0+pow(2,0)*0;

      }

      if(strcmp(mode,"Survival") == 0){

                  if (i<=18)

      buffer[1]=pow(2,5)*1+pow(2,4)*0+pow(2,3)*1+pow(2,2)*0+pow(
2,1)*1+pow(2,0)*1;
                  else

      buffer[1]=pow(2,5)*1+pow(2,4)*0+pow(2,3)*1+pow(2,2)*0+pow(
2,1)*0+pow(2,0)*1;

         }

      if(strcmp(mode,"Nominal") == 0){

            if (i<=18)

      buffer[1]=pow(2,5)*1+pow(2,4)*1+pow(2,3)*1+pow(2,2)*1+pow(
2,1)*1+pow(2,0)*1;
            else

      buffer[1]=pow(2,5)*1+pow(2,4)*1+pow(2,3)*1+pow(2,2)*1+pow(
2,1)*0+pow(2,0)*1;

         }



      int length = 2;            //<<< Number of bytes to write
      if (write(i2c_handler, buffer, length) != length)
      //write() returns the number of bytes actually written, if
it doesn't match then an error occurred (e.g. no response from
the device)
      {
            /* ERROR HANDLING: i2c transaction failed */
            printf("Failed to write to the i2c bus.\n");
      }
      else{
            usleep(500000);
      }


}
```

```
void register_log(eps_hk_t * hk, int file_i2c, char
file_name[30], char mode[30]){

      float i = 0;
      int cont = 45; //We want to execute a 45 min test,
registering every 30 s

      while(i < 2*cont){

            SetOutput(file_i2c,mode,i);

            printf("Elapsed Time: %f minutes.\n",i/2);

            //We execute the get command
            if (GomGetTelemetry(file_i2c, hk) == 0) {
                  eps_hk_print(hk);
            }

            //Just in case there is something wrong
            else{
                  printf("Something went wrong, bruh");
            }

            //We execute the log command
            customed_file_log(hk,file_name);

            sleep(30);

            i++;
      }
}

void main(){
      int file_i2c;
      int length;
      eps_hk_t hk;
      int start=1;
      char command[256];
      char file_name[30];
      char mode[30];

      // Init I2C
      //----- OPEN THE I2C BUS -----
      //The i2c_file opens the i2C file to read and write
between the EPS and the OBC
      char *filename = (char*)"/dev/i2c-1";
      if ((file_i2c = open(filename, O_RDWR)) < 0) {
            //ERROR HANDLING: you can check errno to see what
went wrong
            printf("Failed to open the i2c bus");
            return;
      }

      int addr = 0x02;          //<<<<<The I2C address of the
slave
```

```
      if (ioctl(file_i2c, I2C_SLAVE, addr) < 0) {
            printf("Failed to acquire bus access and/or talk to
slave.\n");
            //ERROR HANDLING; you can check errno to see what
went wrong
            return;
      }

      while(start==1){
            printf("Command: ");
            if (fgets(command, sizeof(command), stdin) != NULL) {
                  if (strncmp(command, "get", strlen("get")) == 0)
{
                        // they are equal so, respond get
                        if (GomGetTelemetry(file_i2c, &hk) == 0) {
                              eps_hk_print(&hk);
                        }
                  }
                  if (strncmp(command, "set", strlen("set")) == 0)
{//We compare the command with the set string but only the set
chars, not the whole command.
                        // they are equal so, respond set
                        int id, state;
                        sscanf(command, "set %d %d\n", &id,
&state);
                        printf("Setting PoL %d to %d\n", id,
state);
                        SetOutputStatus(file_i2c,id,state,&hk);
                  }
                  if (strncmp(command, "log", strlen("log")) == 0)
{
                        // they are equal so, respond log
                        file_log(&hk);
                  }
                  if (strncmp(command, "exit", strlen("exit")) ==
0) {
                        // they are equal so, respond exit
                        start = 0;
                  }
                  if (strncmp(command, "register",
strlen("register")) == 0) {
                        // they are equal so, respond register
                        sscanf(command, "register %s %s\n",
&file_name, &mode);
                        printf("Starting the 45 min test with a
record every 30 s in the log file %s in %s mode\n",
file_name,mode);
                        if((strcmp(mode,"NoPoL") == 0) ||
(strcmp(mode,"Survival") == 0) || (strcmp(mode,"Nominal") ==
0)){
                              register_log(&hk, file_i2c,
file_name,mode);
                        }
                        else{
                              printf("You have written the mode
name wrong, u dumbass.");
```

```
                        }
                    }

                    //returnData(command,hk,&start);
                } else {
                    start = 0;
                }
            }
        }
```

## 8.4. EPS Caracterization

NANOSAT LAB

# Characterization Test of the Subsystem's performance

*Electrical Power Subsystem*

*University Name:* Universitat Politècnica de Catalunya (UPC)
Campus Nord, building D3
08034 Barcelona, SPAIN
Date: 11/07/2020
*Reference:*
*Distribution List:*

| Destination | Contact |
|---|---|
| [3]Cat-4 Team | 3cat4@tsc.upc.edu |

# Authors


| First Name | Last Name | Contact | Role |
|---|---|---|---|
| Juanjo | Medina Musellas | juanjomedinamusellas@gmail.com | --- |
| Albert | Rodríguez Casellas | albert.rodriguez.casellas@gmail.com | --- |


# Revised by


| First Name | Last Name | Contact | Role |
|---|---|---|---|
| Lara | Fernandez Capon | lara-pilar.fernandez@tsc.upc.edu | System Engineer |

# Table of Contents

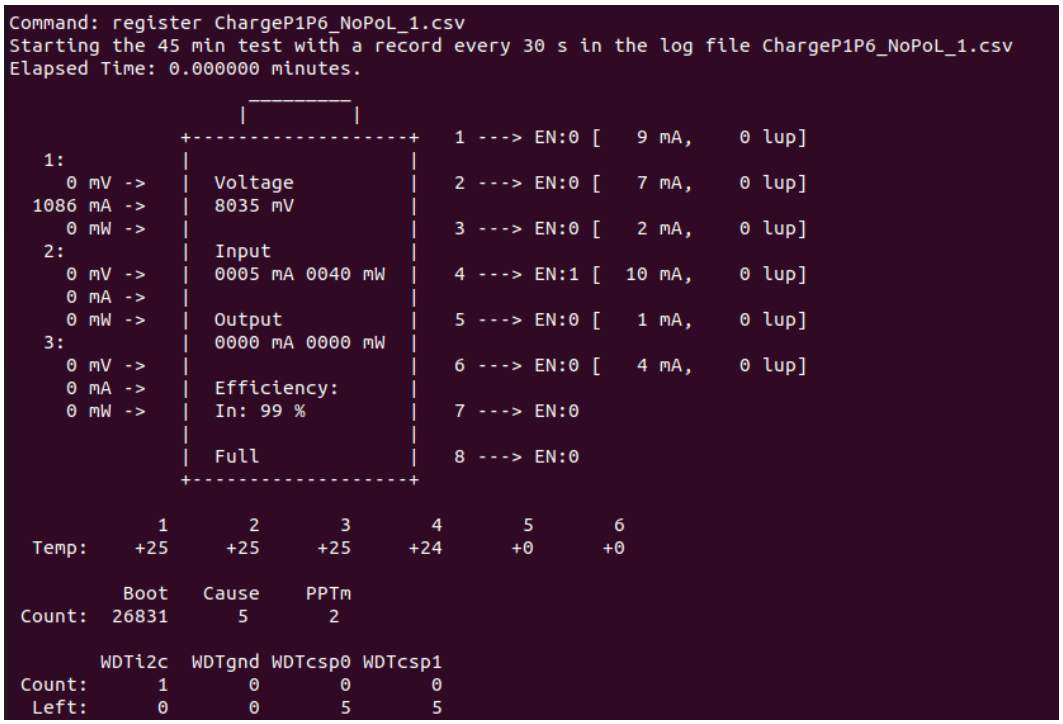
## Document Scope

This document aims to explain the results obtained in the Characterization Test carried out within the Electrical Power Subsystem (EPS).

The Characterization Test will study the EPS performance when the subsystem is subjected to various test conditions.

The Characterization Test will consist in a determined number of tests, consisting (generally) each of them in a 45 minute test, gathering information about the subsystem status every 30 seconds. The information will be collected through a C Script, which runs in the Raspberry (that effectuates the OBC function).

This is an example of the collected measurements in one time slot.




The Characterization Test will study every situation that the EPS could possibly face. In order to study some real cases, we are going to test the subsystem under the 3CAT4 mission modes. These modes show different configurations of the subsystems so that they can carry a determined task. The modes are:

1- No Subsystem connected (every Point of Load (PoL) disconnected). Although this mode is not really one of the mission modes, we consider it sufficiently important to study it.

2- Released Mode.

3- Pre-detumbling Mode.

4- Detumbling Mode.

5- Detumbled (Sun - Safe) Mode.

6- Survival Mode

7- Nominal Mode

We are going to repeat the test of each mode in different situations so as to understand the different cases and performances that the EPS could encounter. The situations are:

Situation A: Raspberry (OBC subsystem) connected and EPS connected

Situation B: Raspberry connected and EPS disconnected

Situation C: Raspberry disconnected and EPS connected

Situation D: Raspberry disconnected and EPS disconnected

# SITUATION A: Raspberry (OBC subsystem) connected and EPS connected

In this case, we are going to keep both EPS and OBC (Raspberry) switched ON. This situation is going to be tested under every single mission mode, specified in the Document Scope.

The following table shows the different active components in each mode (we have not considered the AOCS Subsystem because it was not properly specified in the power budget when we did the first approach).

| | PoL 1: COMMS | PoL 2: Payload | PoL 3: MTQ | PoL 4: OBC | PoL 5: UHF/AIS and Helix | PoL 6: AOCS |
|---|---|---|---|---|---|---|
| No PoL | | | | | | |
| Released Mode | | | | X | X | |
| Pre-detumbling Mode | X | | | X | X | X |
| Detumbling Mode | X | | X | X | X | X |
| Detumbled (Sun-Safe) Mode | X | | X | X | X | X |
| Survival Mode | X | | | X | | X |
| Nominal Mode | X | X | X | X | X | X |

The following table shows the tests that are going to be carried out in Situation A.

| | DISCHARGE MODE | P8 CHARGE MODE | P1-P6 CHARGE MODE |
|---|---|---|---|
| No PoL | X | X | X |
| Released Mode | X | X | X |
| Pre-Detumbling Mode | | | |
| Detumbling Mode | | | |
| Detumbled (Sun - Safe) Mode | | | |
| Survival Mode | | | |
| Nominal Mode | X | X | X |

## A.1- No PoL Mode

This test is going to be done without any load connected to the Point of Loads, and hence, the only fixed connections will be the I2C and Ground ports between the EPS and the Raspberry so as to be able to get the information about the EPS state.

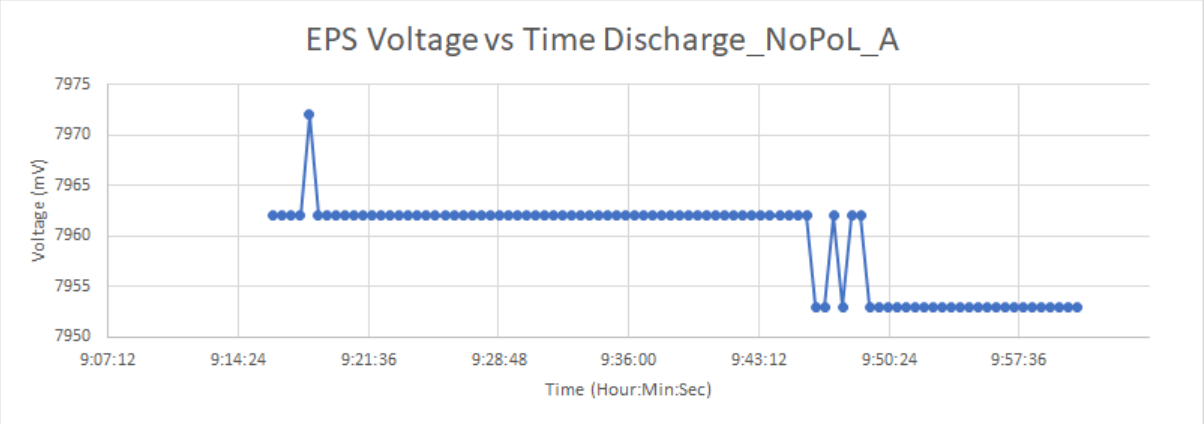
**Observation:** *We believe that there are two charge effects in the whole circuit. One of them is related to the own EPS circuit and the other one is related to the Raspberry's connections. If we charge the EPS, with the EPS in mode ON and with a 5V/1A input we obtain 2.6V/1A (the power supply is limiting the current, but the datasheet states a 3.7V/1A with everything connected). However, with the EPS and the Raspberry connected, with the same input, we* ***also*** *get 2.6V/1A (the same value as before). Nevertheless, if we connect the RBF (EPS mode OFF) with the Raspberry connected, sometimes the input value is 3.3V/1A and sometimes 2.6V/1A (also the same value as before). It is somehow weird to get 2.6V with either the EPS connected or the EPS and the Raspberry connected, and also we can not understand the random change in the input voltage when the EPS is in mode OFF and the Raspberry connected.*

### A.1.1- Discharge Mode

In this mode, we have the EPS not charging, so there are no connections between the EPS and the Power Supply.
The following plots study the EPS performance under these conditions.

- **Voltage plot.**

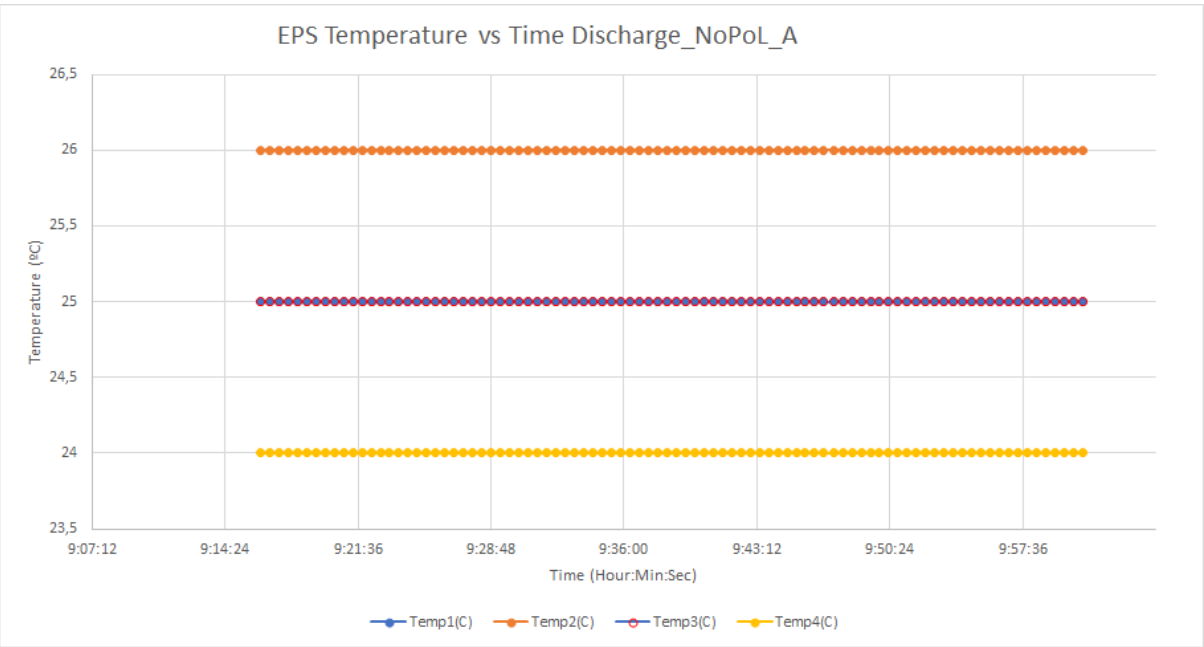
This plot shows how the voltage change with time influence.




As it is done in Discharge Mode, it is correct to see a continuous decrease in the voltage stored within the batteries. Nevertheless, as the EPS is the only subsystem connected, we see no decrease in this case.

- **Temperature plot.**

This plot studies the temperature change of every single temperature sensor as a function of time.

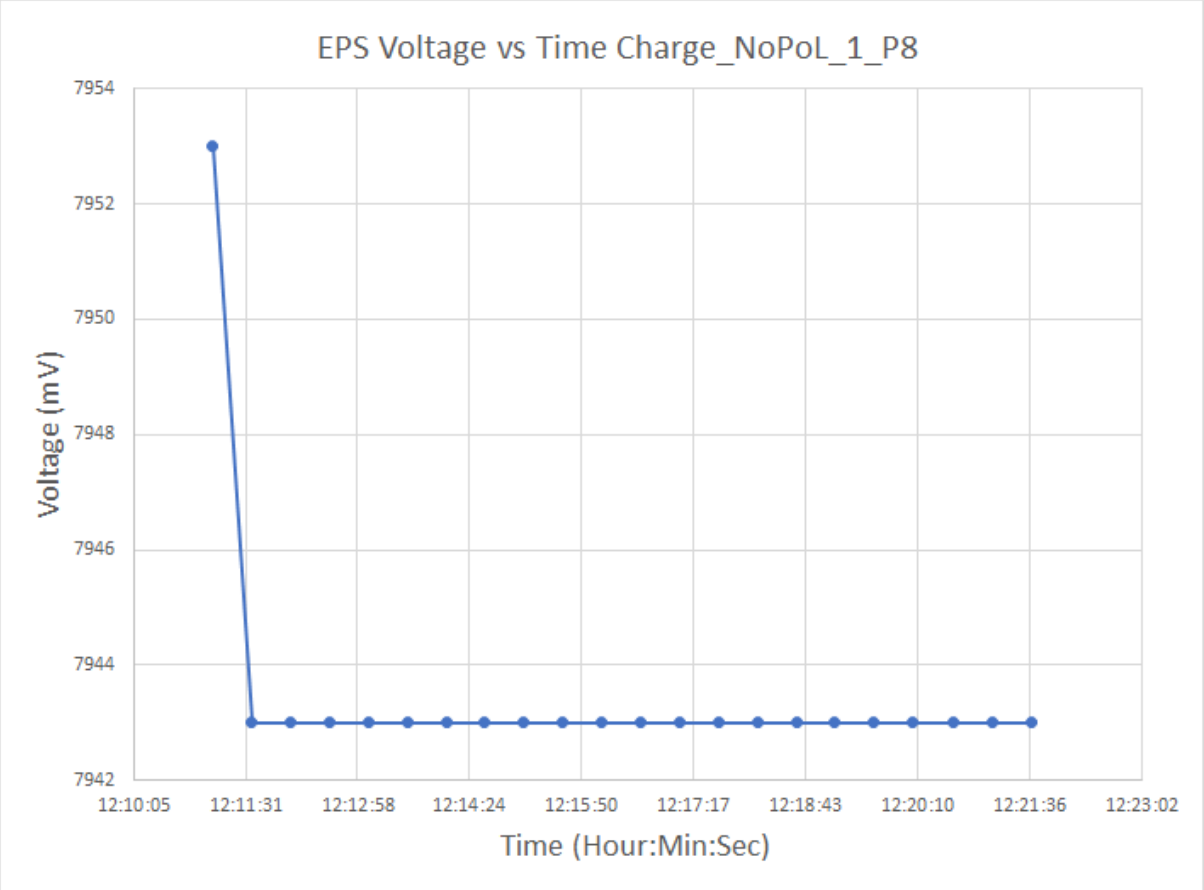


In this case, there is practically no effect on the Temperature in all the EPS sensors.

### A.1.2- P8 Charge Mode

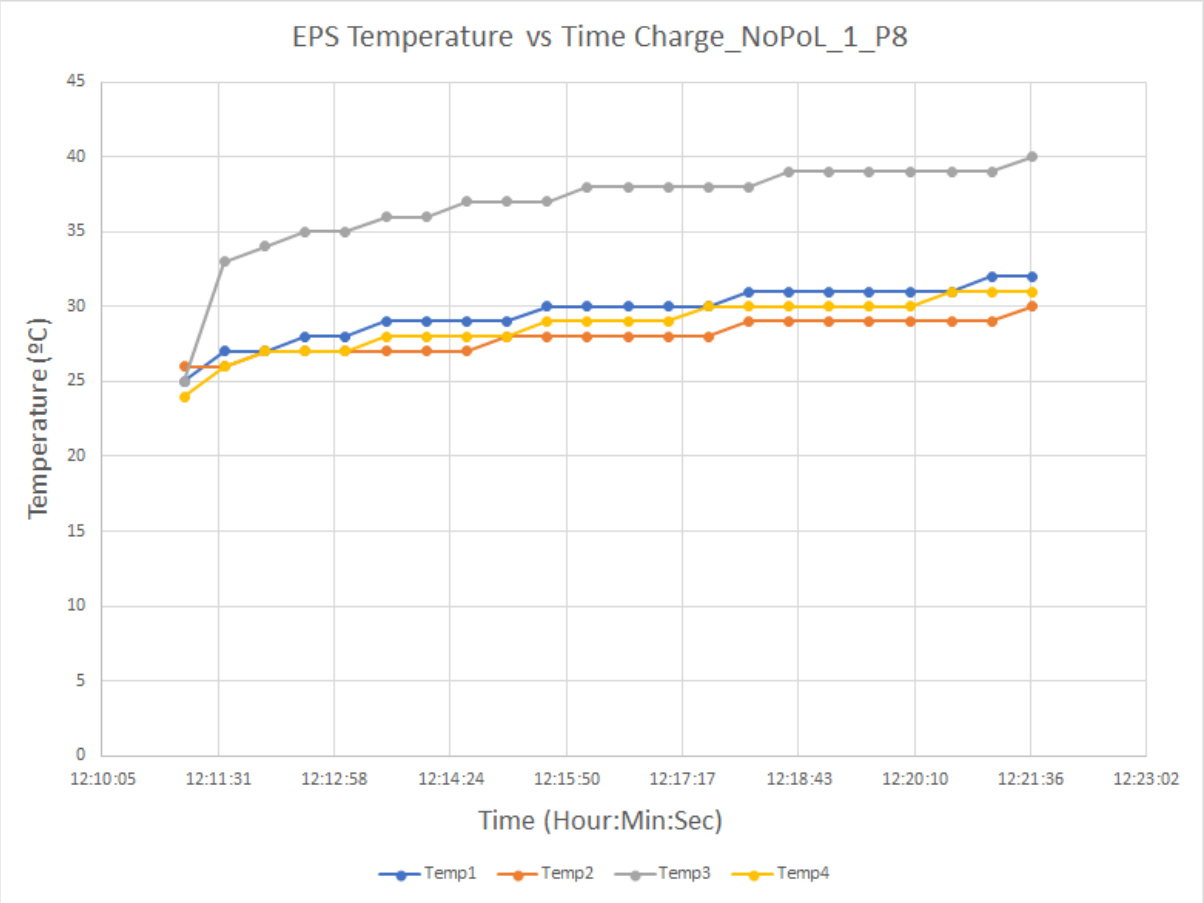
In this mode we have the EPS connected to the Power Supply (which theoretically will be charging the EPS) by the P8 pin, using the GND and the CHARGE pins inside the P8.
This test has not been finished due to the huge Temperature increase. We have decided to establish the 40ºC as a limit for any test.

- **Voltage plot**




In this mode, the voltage should be increasing owing to the fact that it is a charge mode. However, the voltage is decreased to from 7.953V to 7.943V, and then the value is kept constant.

- **Temperature plot.**

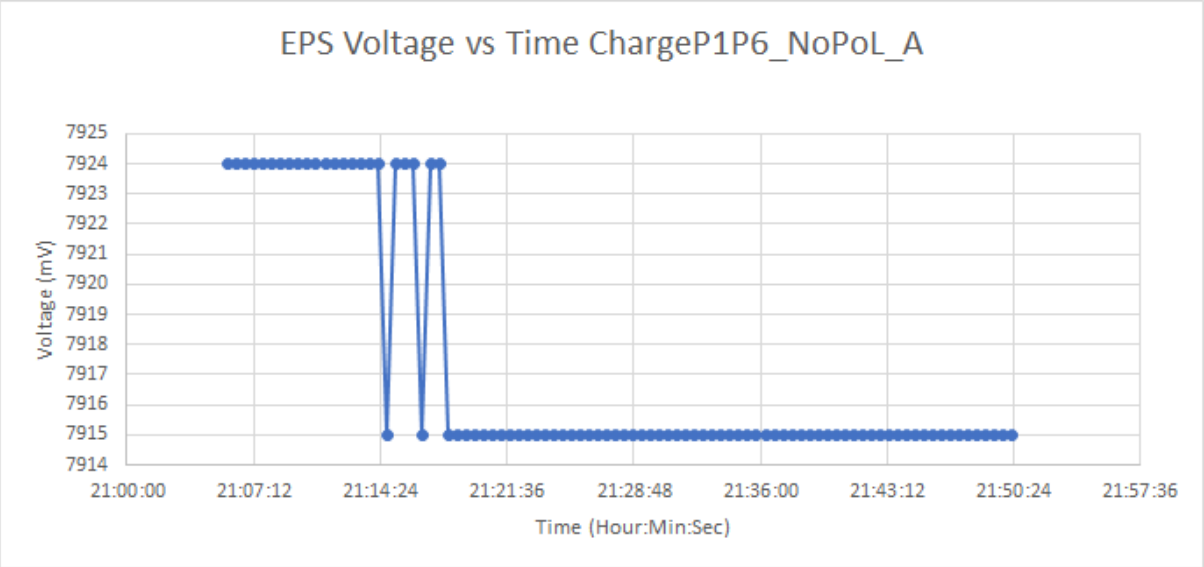



As for the Temperature, the Temperature sensors suffer a great increase of their values, which is not acceptable. It could be caused by the charge effects mentioned previously.

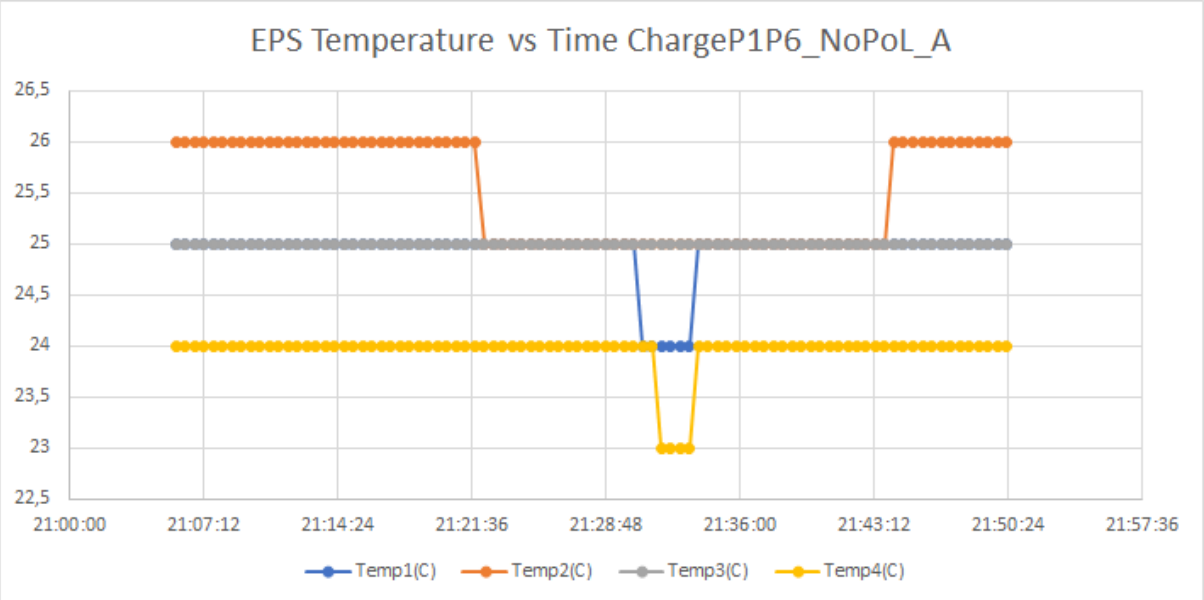
### A.1.3- P1-P6 Charge Mode

In this mode, we have the EPS connected to the Power Supply (which, again, will be charging the EPS) by the Solar-Panel input connections, which are the P1 to P6 pins (in this case, the connection is done by using the P3 pin).

- **Voltage plot.**



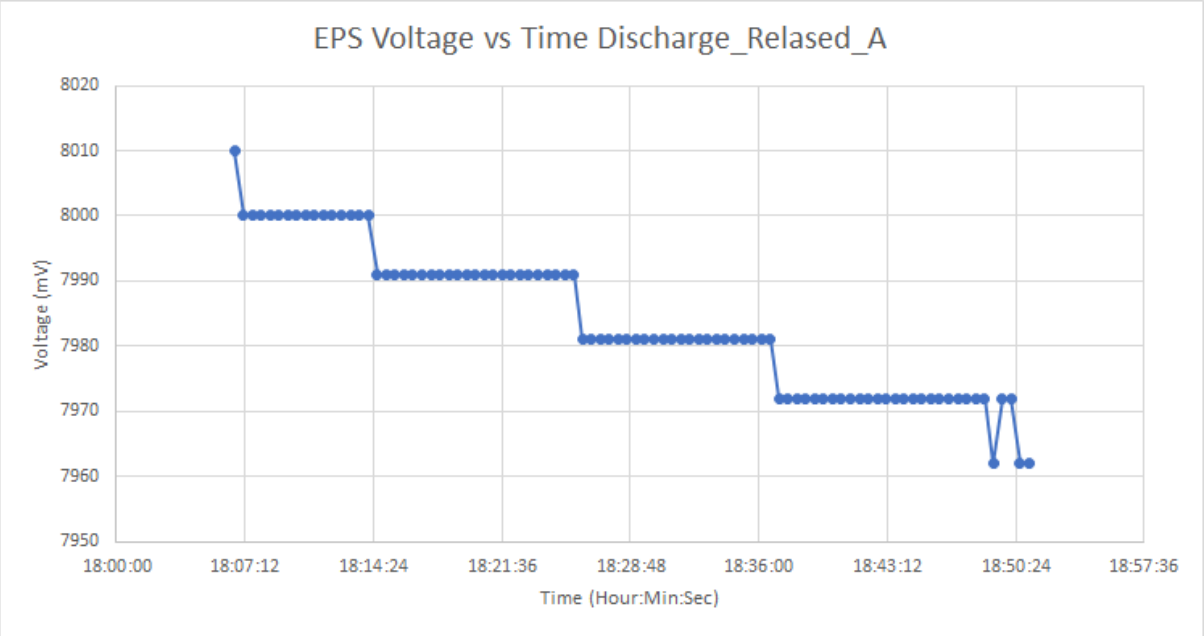

- **Temperature plot.**



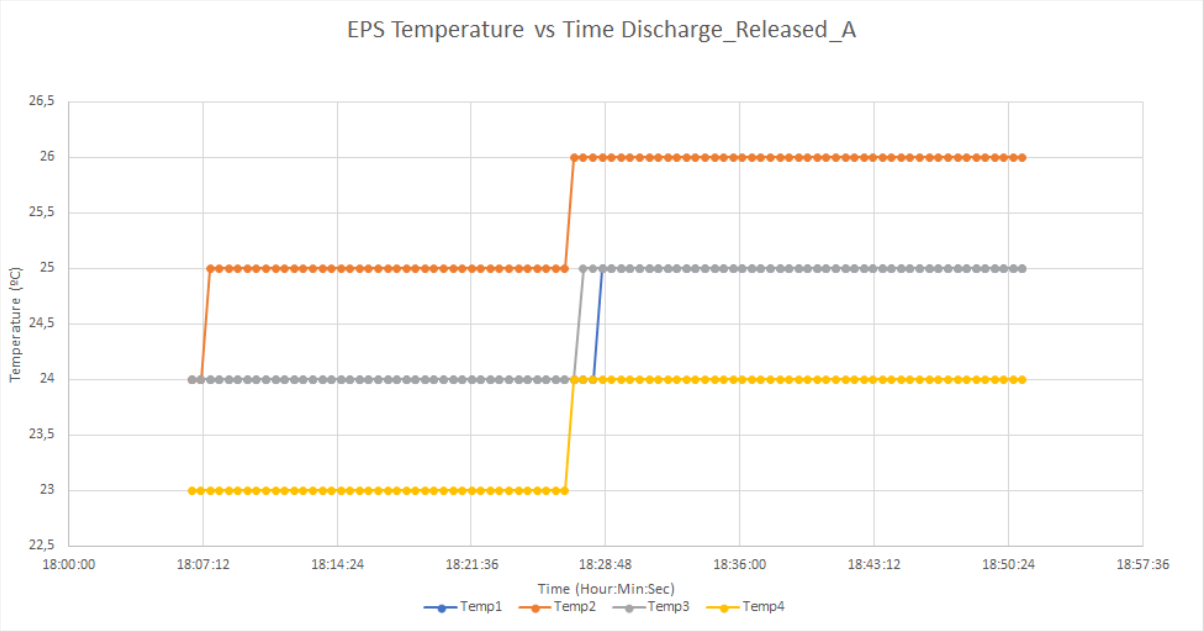
# A.2- Released Mode

## A.2.1- Discharge Mode

- **Voltage plot.**

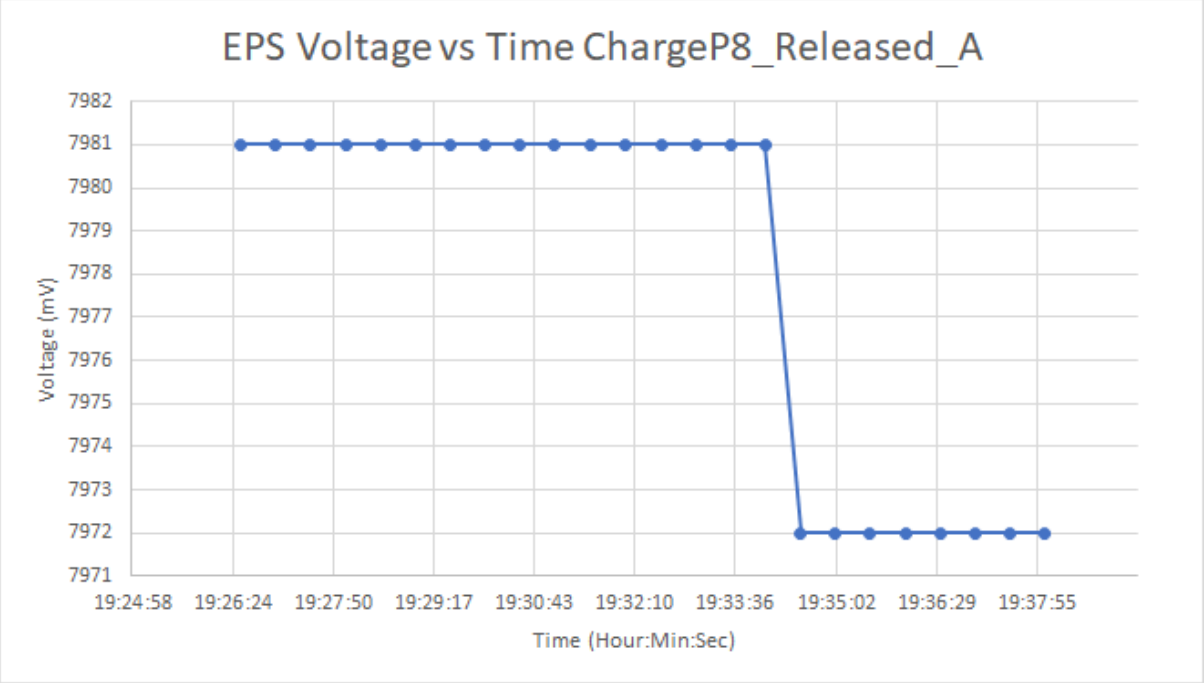



- **Temperature plot.**

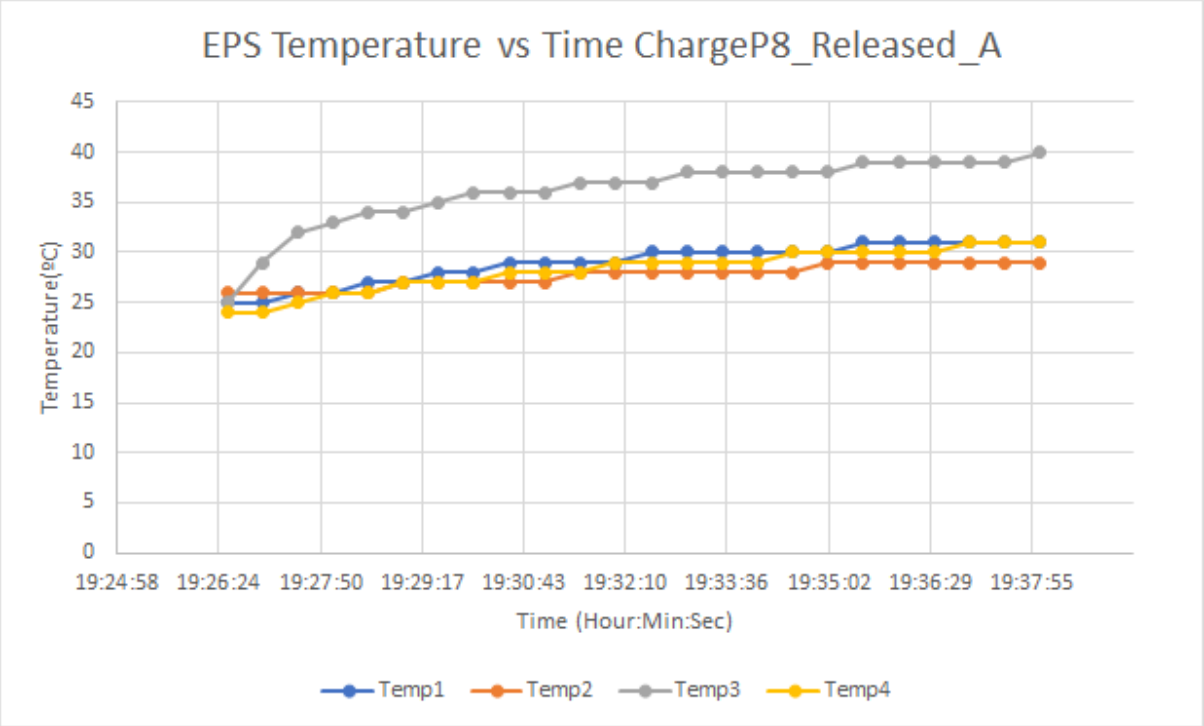


### A.2.2- P8 Charge Mode

- **Voltage plot.**

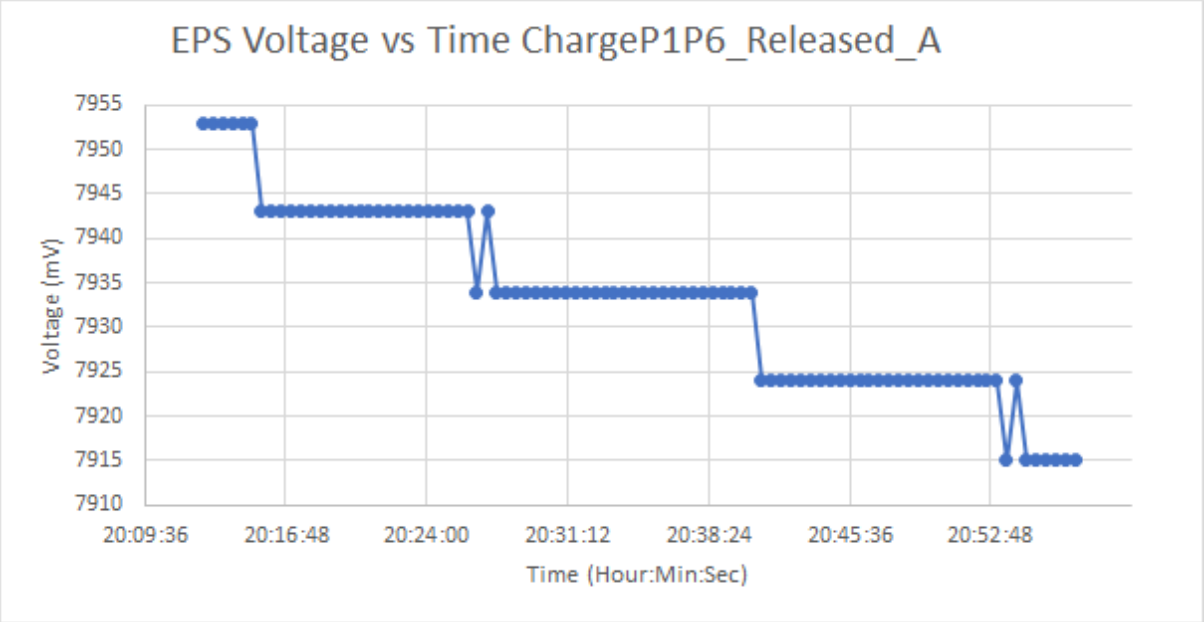



- **Temperature plot.**

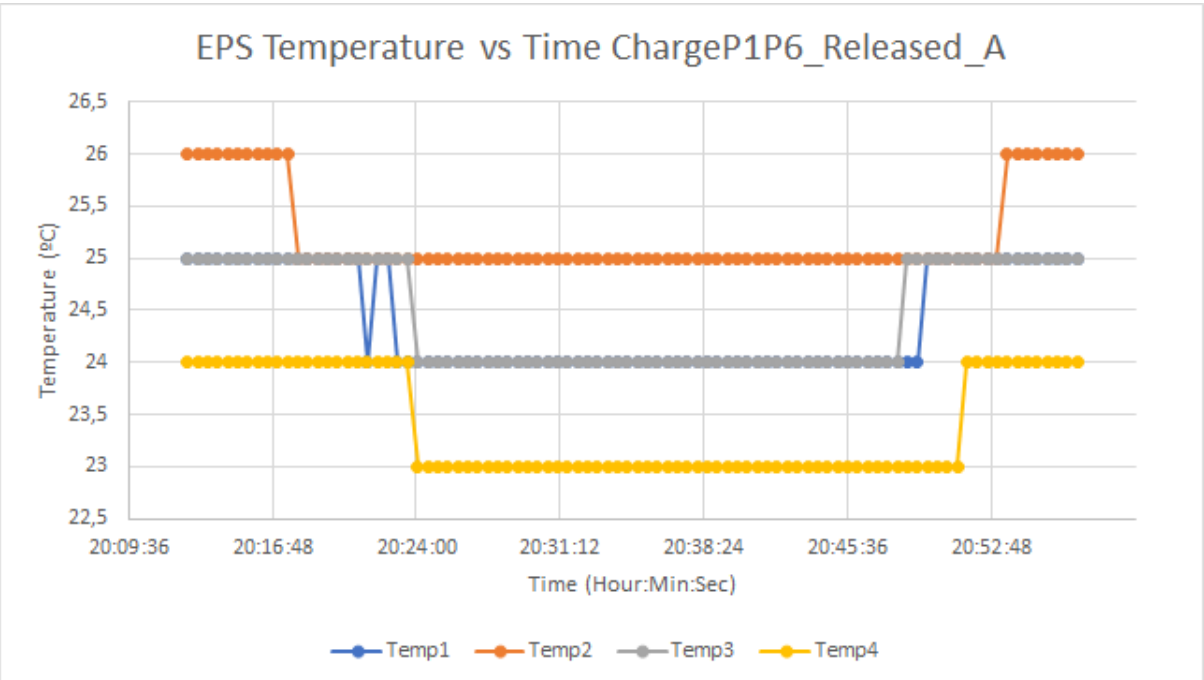


### A.2.3- P1-P6 Charge Mode

- **Voltage plot.**



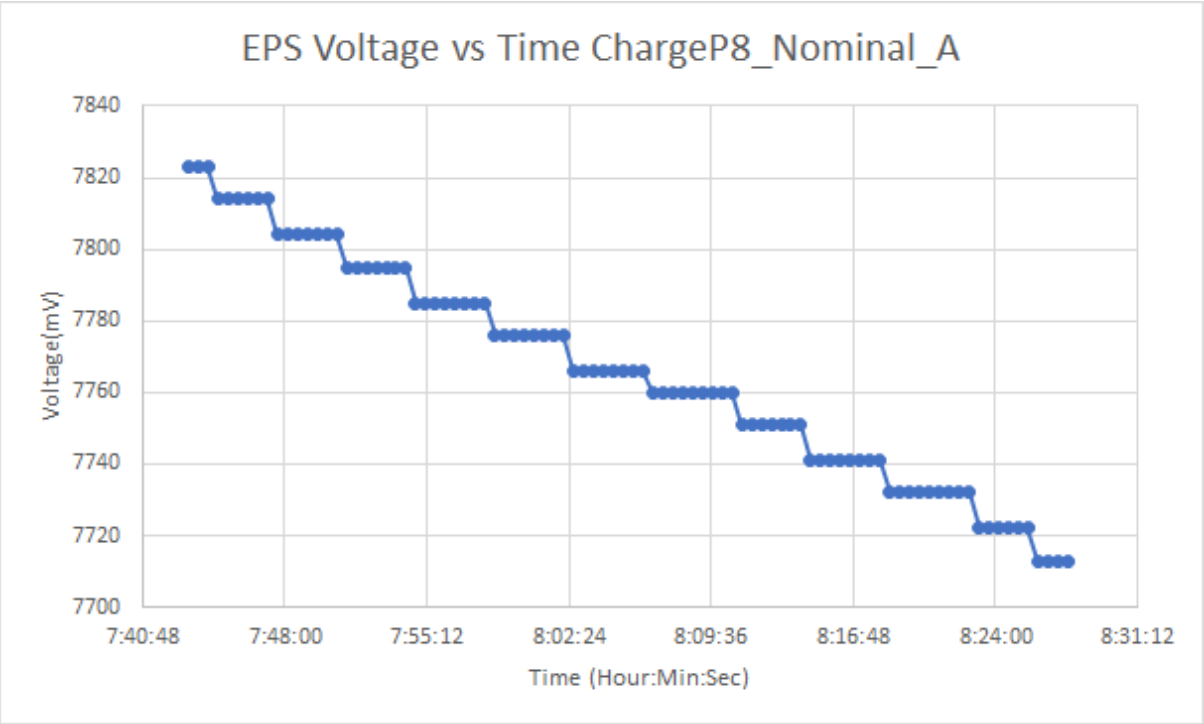

- **Temperature plot.**



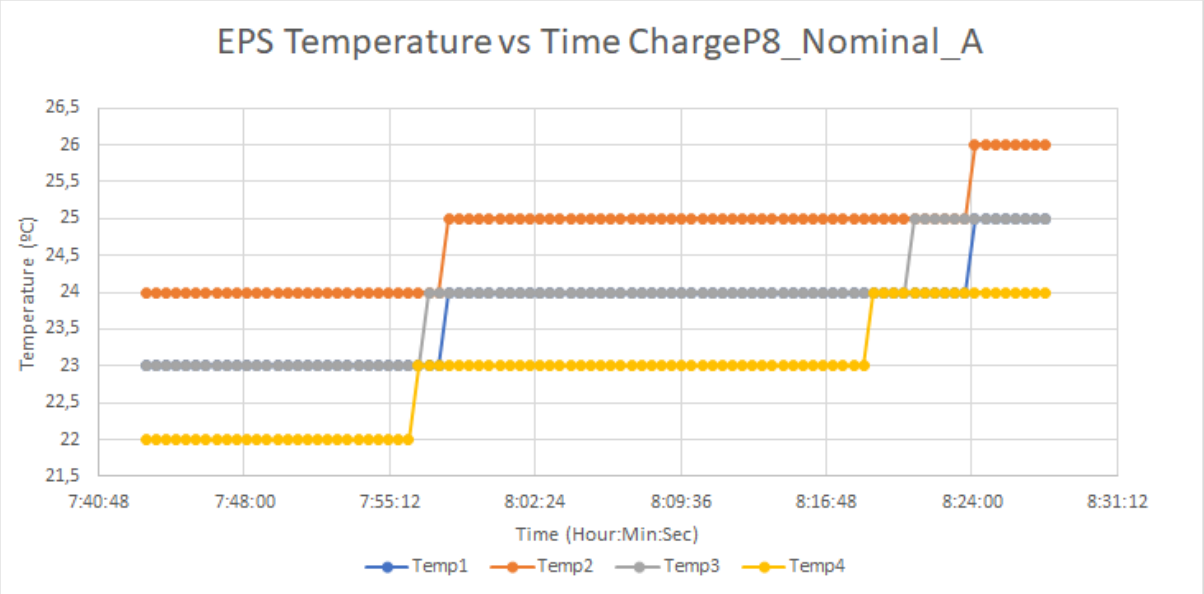
# A.3- Nominal Mode

## A.3.1- Discharge Mode

- **Voltage plot.**

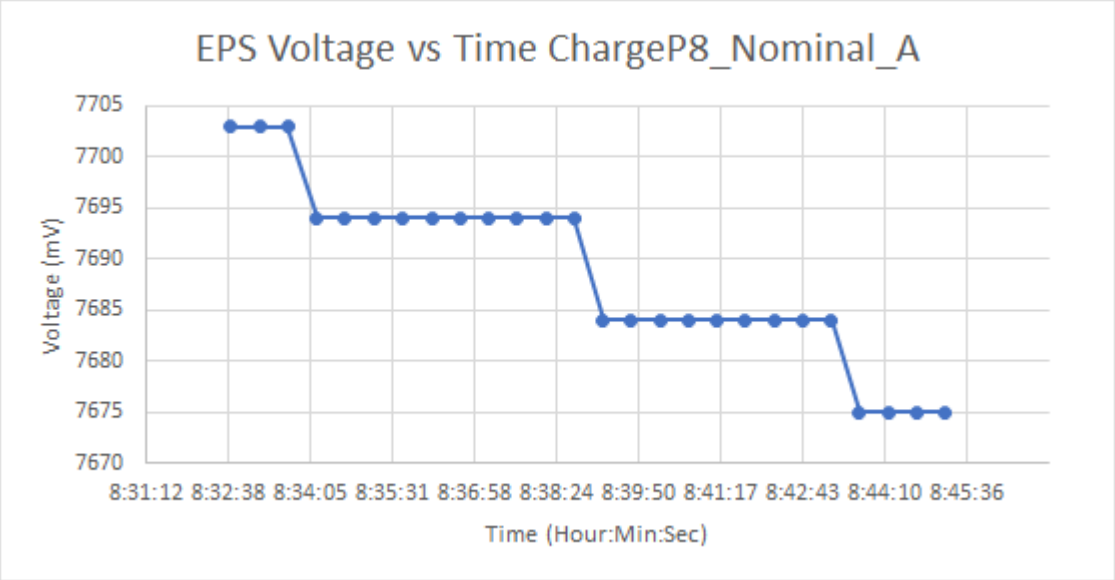



- **Temperature plot.**

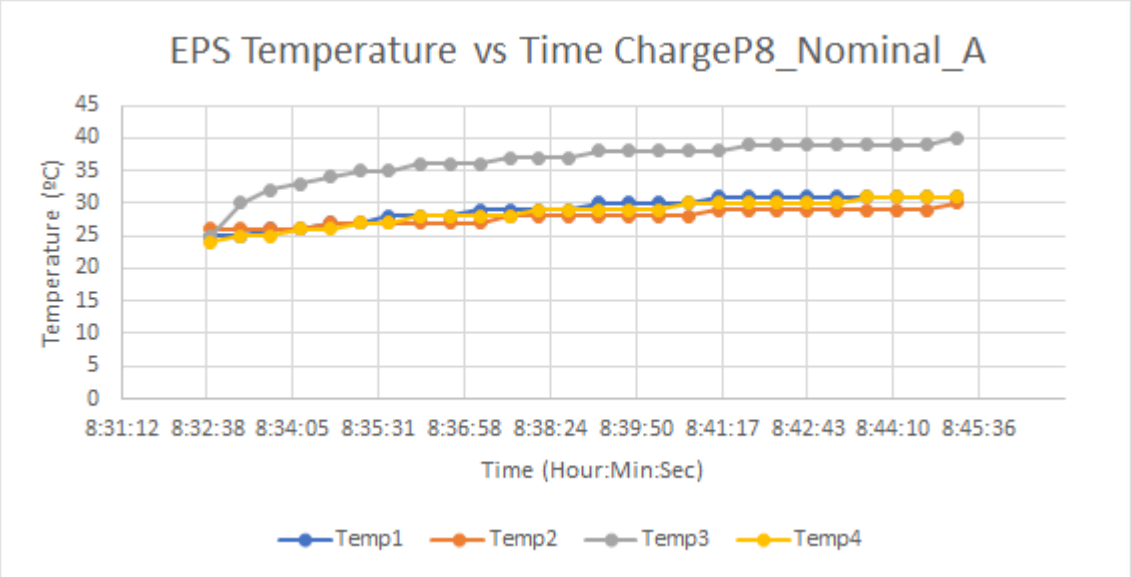


### A.3.2- P8 Charge Mode

- **Voltage plot.**

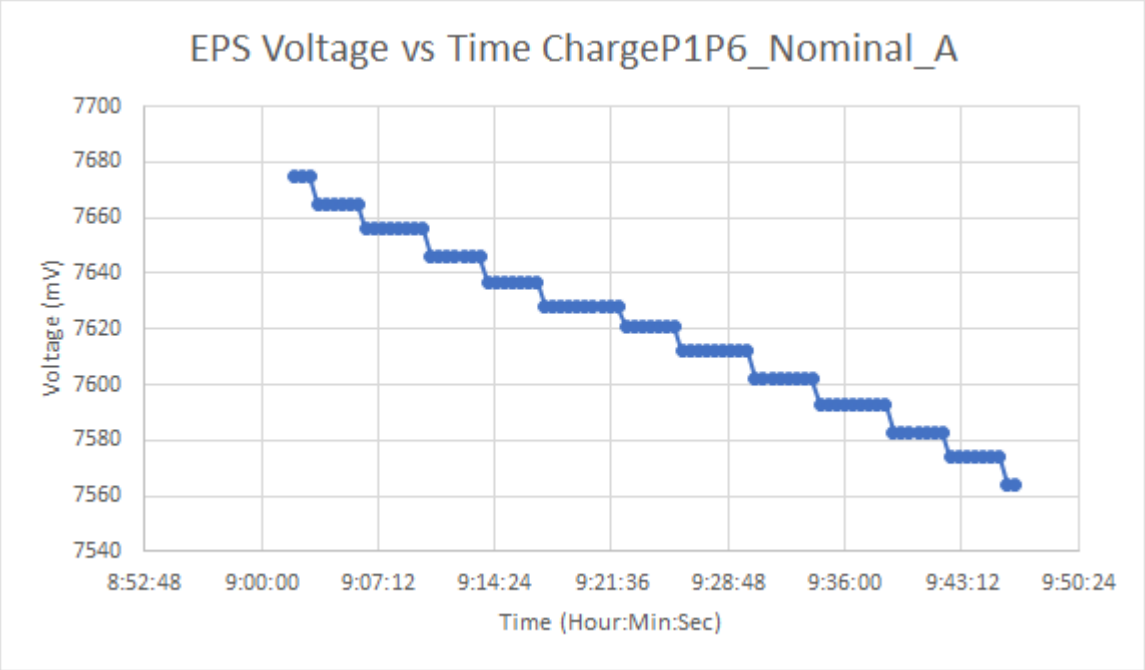



- **Temperature plot.**

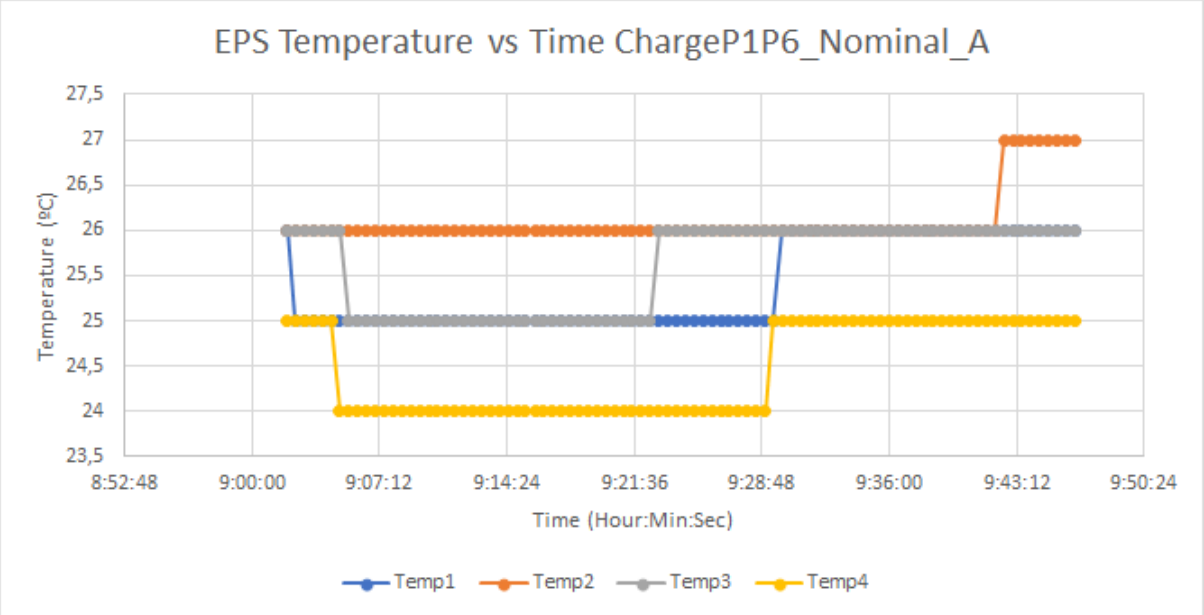


### A.3.3- P1-P6 Charge Mode

- **Voltage plot.**




- **Temperature plot.**



# SITUATION B: Raspberry connected and EPS disconnected

In this case, we are going to keep the EPS switched OFF and the Raspberry switched ON.
In this case, as the EPS is switched OFF, in order to collect the information, we will have to switch it ON every 30 seconds to obtain the data, and then switch the EPS OFF again. As the EPS will remain switched off, the only possible tests are the ones done in No PoL Mode.

Thus, the following table shows the tests that are going to be carried out in Situation B.

| | DISCHARGE MODE | P8 CHARGE MODE | P1-P6 CHARGE MODE |
|---|---|---|---|
| No PoL | | X | X |
| Released Mode | | | |
| Pre-Detumbling Mode | | | |
| Detumbling Mode | | | |
| Detumbled (Sun - Safe) Mode | | | |
| Survival Mode | | | |
| Nominal Mode | | | |

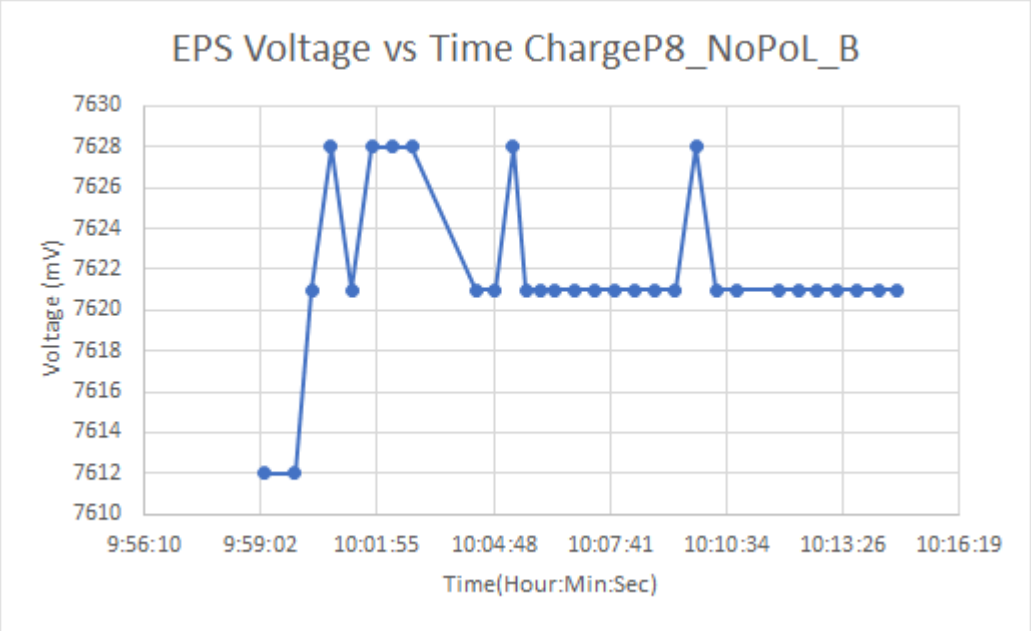
## B.1- No PoL Mode

### B.1.1- Discharge Mode

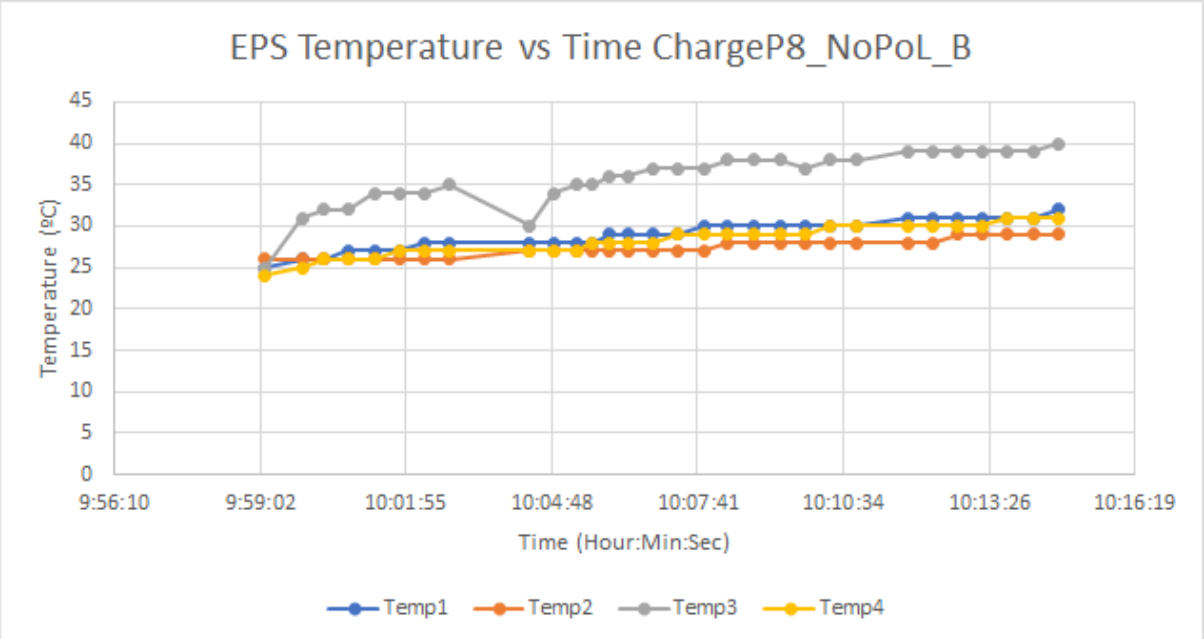
As the EPS is actually disconnected, there can be no discharge mode.

### B.1.2- P8 Charge Mode

- **Voltage plot**



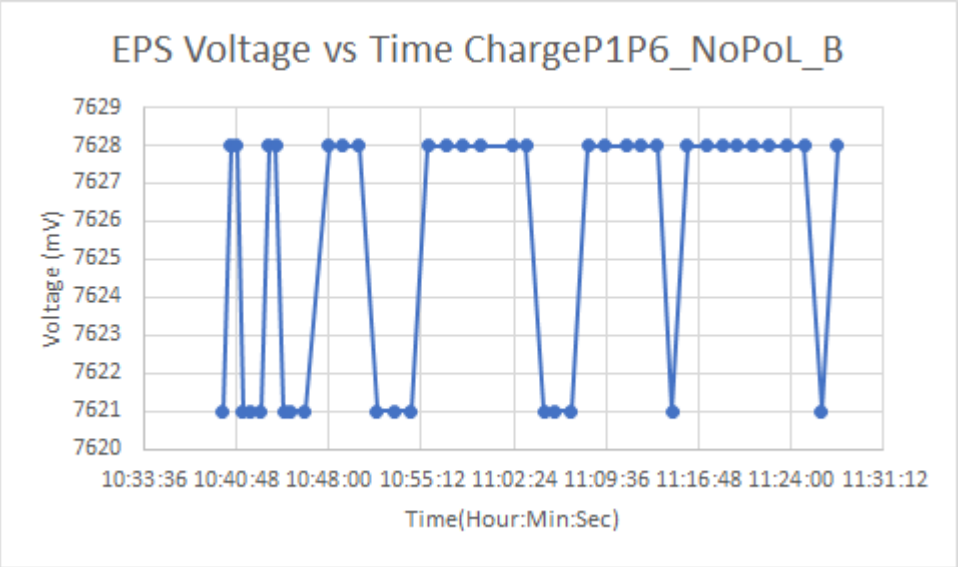

- **Temperature plot.**



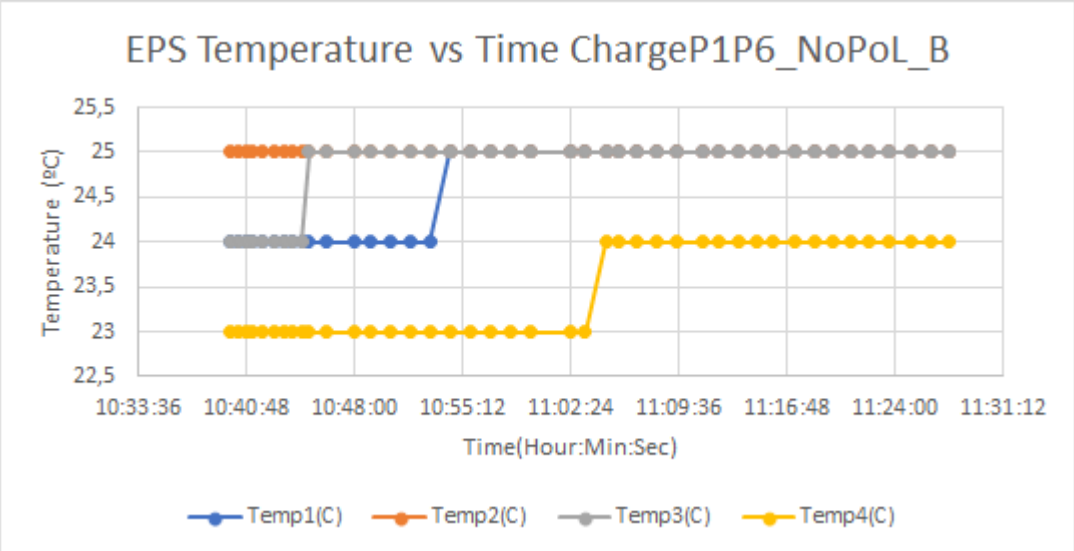
### B.1.3- P1-P6 Charge Mode

**Observation:** *When we try to use the P1 (from P1 to P6) to charge the EPS, if we maintain the EPS switched OFF, it appears in the Power Supply a value of 5V/0A, so we assume it is not charging the subsystem. If we connect the Raspberry, there is no change in the Power Supply input values. However, if we switch ON the EPS, the Power Supply's input value is 0.3V/1A, so the Power Supply is limiting the current and thus the voltage decreases. It is different from the P8 Charge case, where the Power Supply is limited either switching ON the EPS or the Raspberry.*

- **Voltage plot.**




- **Temperature plot.**



# SITUATION C: Raspberry disconnected and EPS connected

In this case, we are going to keep the EPS switched ON and the Raspberry switched OFF. In this case, as the Raspberry (which is responsible to gather the EPS information) is switched OFF, we will have to switch it ON every 30 seconds to obtain the data, and then switch the Raspberry OFF again. In this case, we can test every single mission mode owing to the fact that the EPS is switched ON.

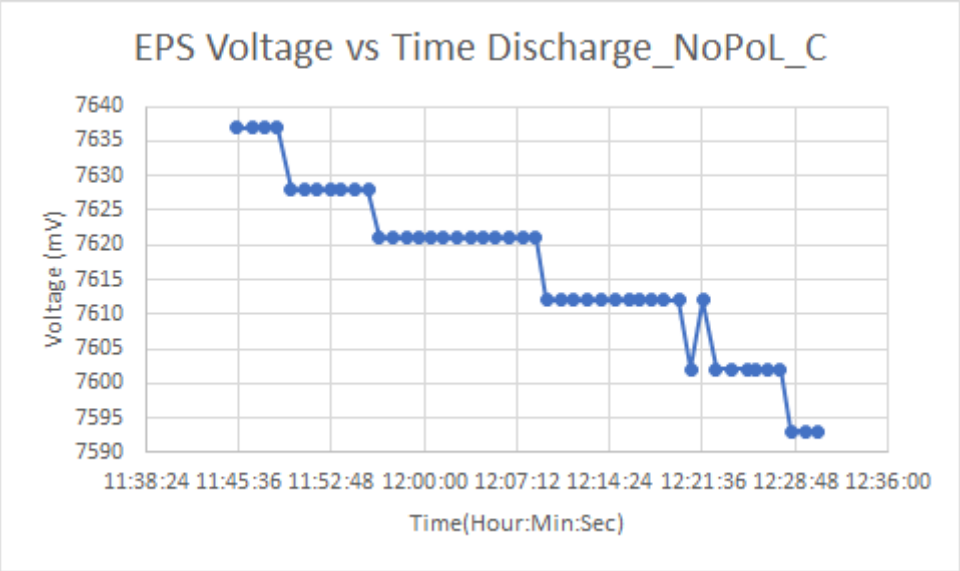
Thus, the following table shows the tests that are going to be carried out in Situation B.

| | DISCHARGE MODE | P8 CHARGE MODE | P1-P6 CHARGE MODE |
|---|---|---|---|
| No PoL | X | X | X |
| Released Mode | | | |
| Pre-Detumbling Mode | | | |
| Detumbling Mode | | | |
| Detumbled (Sun - Safe) Mode | | | |
| Survival Mode | | | |
| Nominal Mode | X | X | |

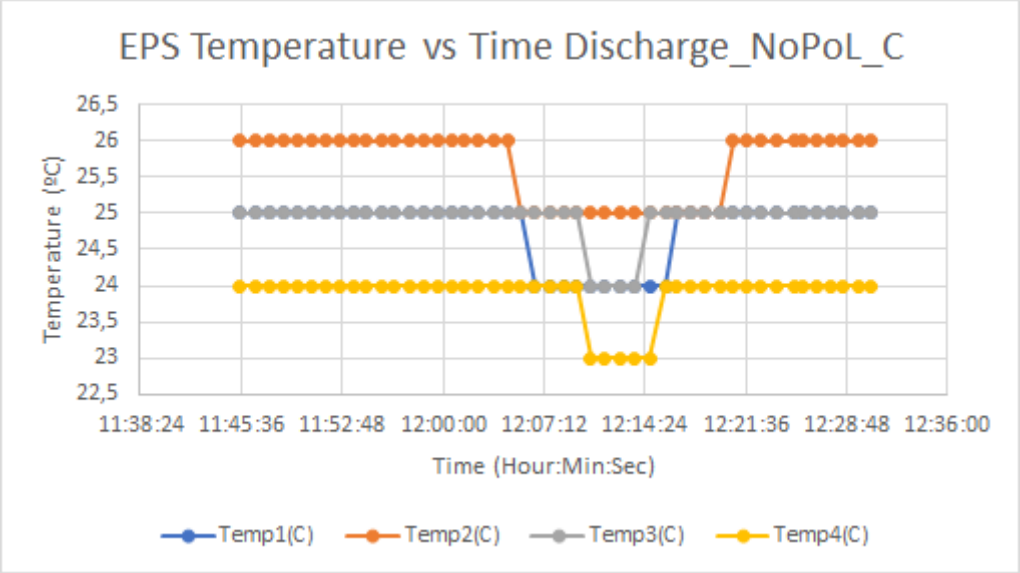
# C.1- No PoL Mode

## C.1.1- Discharge Mode

- **Voltage plot.**

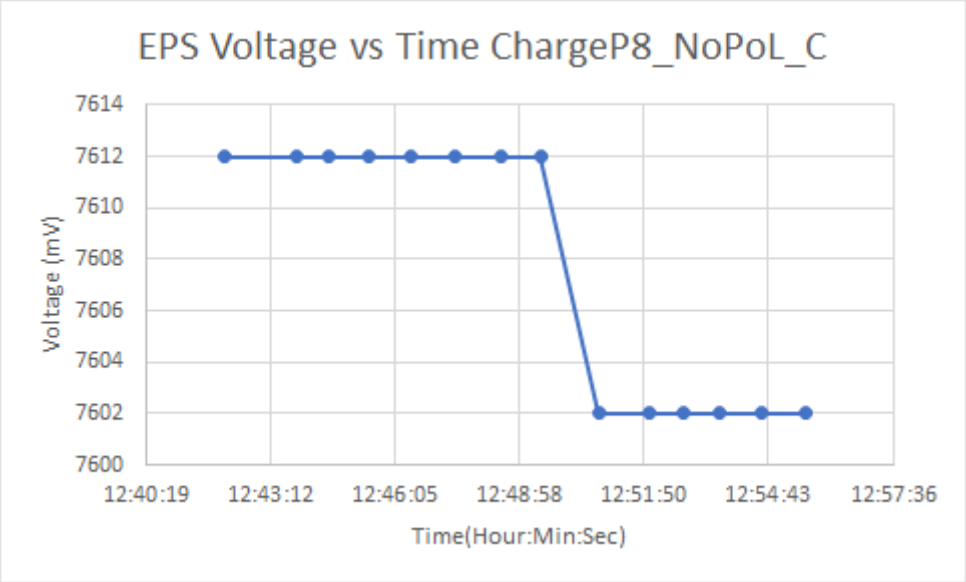



- **Temperature plot.**

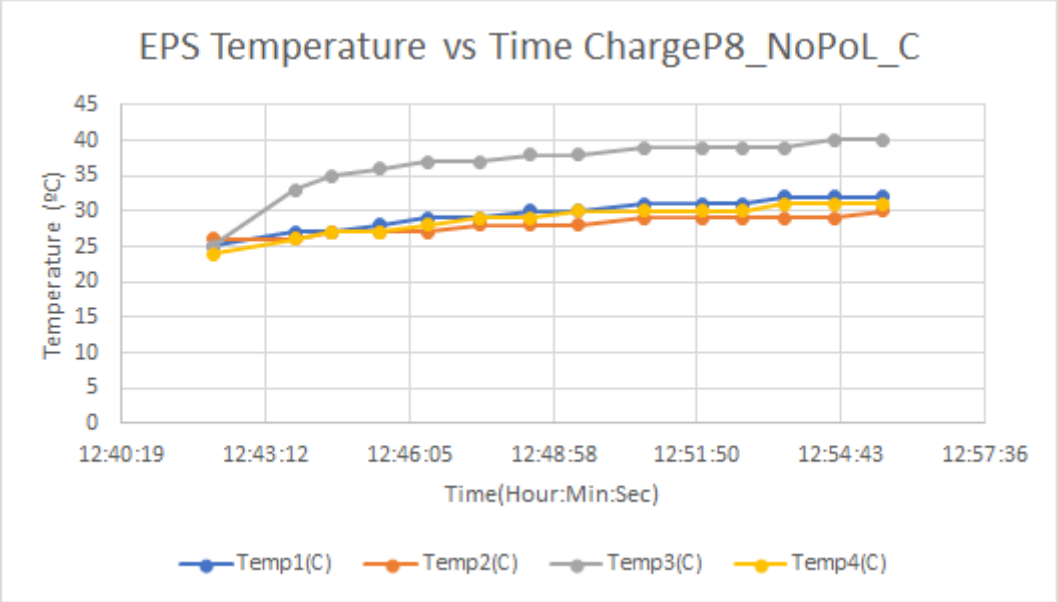


## C.1.2- P8 Charge Mode

- **Voltage plot**

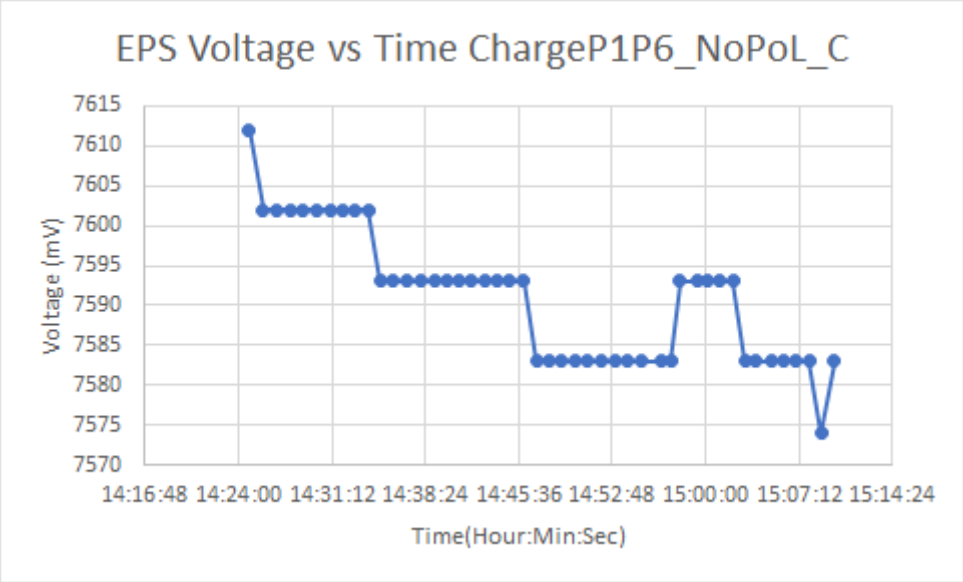



- **Temperature plot.**

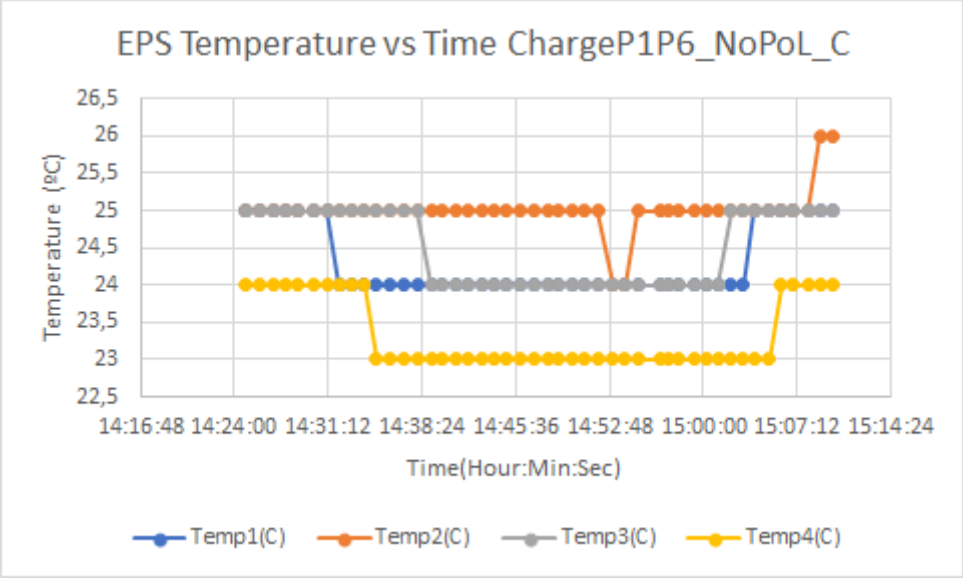


### C.1.3- P1-P6 Charge Mode

- **Voltage plot.**



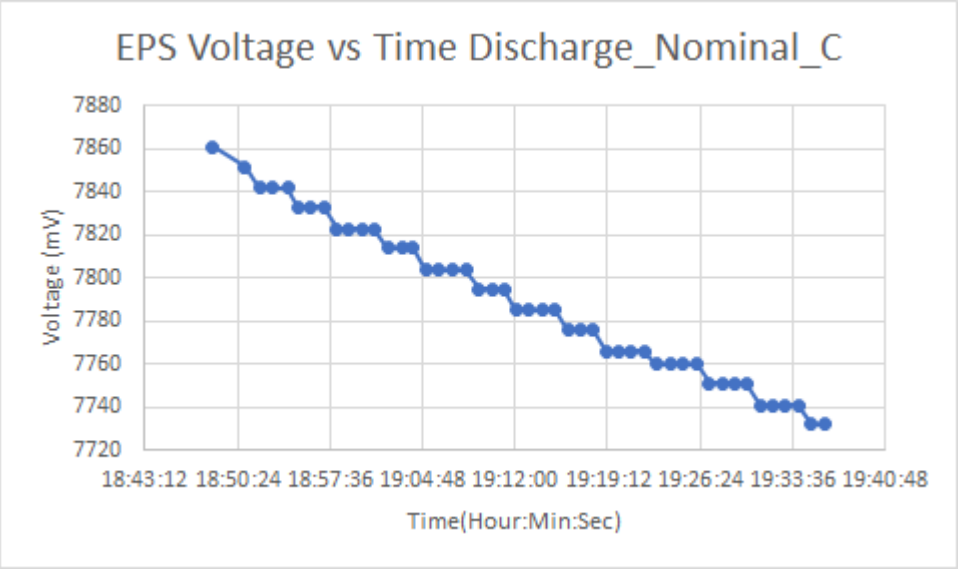

- **Temperature plot.**



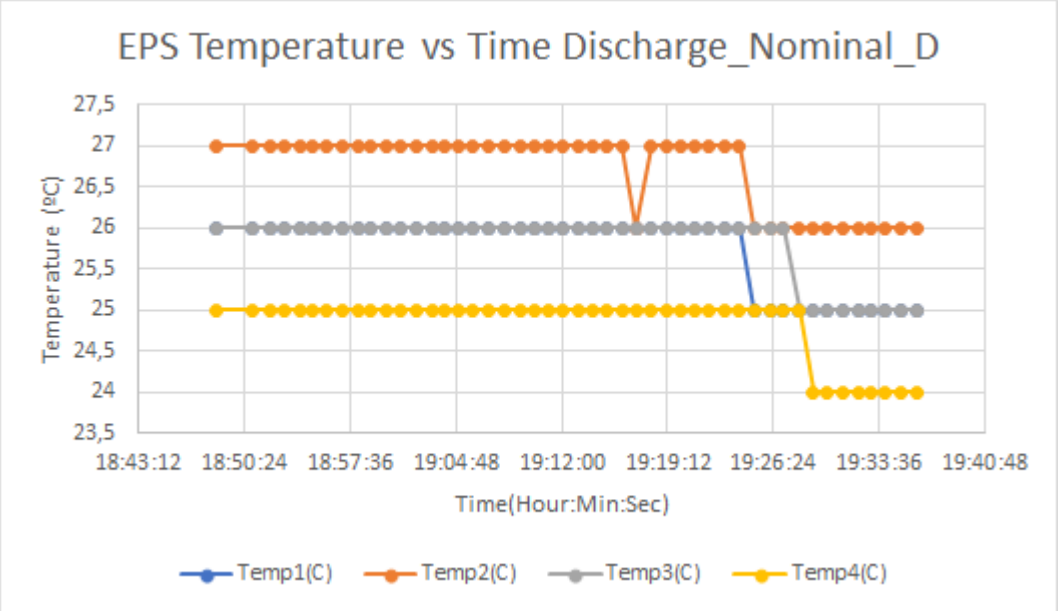
# C.2- Nominal Mode

## C.2.1- Discharge Mode

- **Voltage plot.**

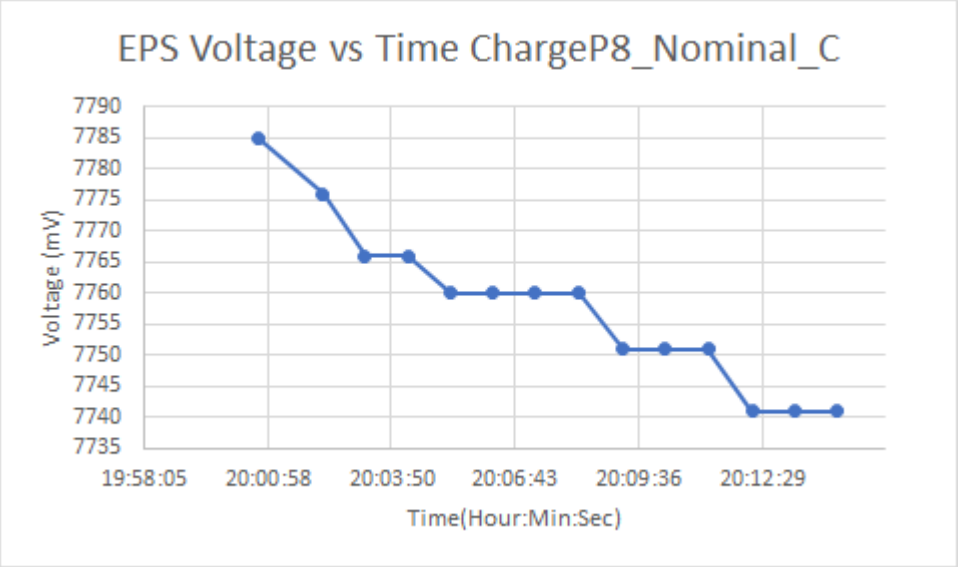



- **Temperature plot.**

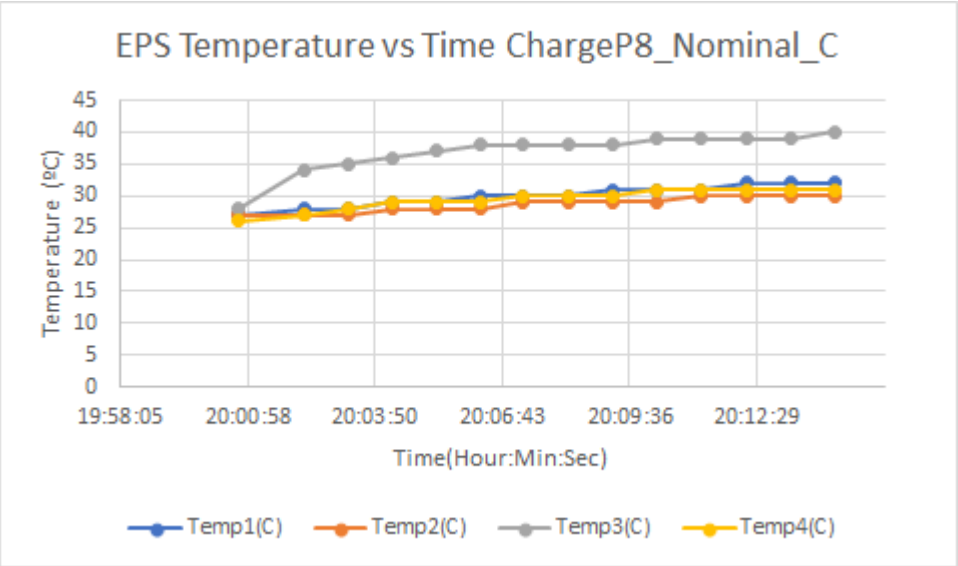


### C.2.2- P8 Charge Mode

- **Voltage plot.**




- **Temperature plot.**



# SITUATION D: Raspberry disconnected and EPS disconnected

In this case, we are going to keep both the EPS and the Raspberry switched OFF. In this case, as the EPS and also the Raspberry are switched OFF, in order to collect the information, we will have to switch them ON every 30 seconds to obtain the data, and then switch the EPS and Raspberry OFF again. As the EPS will remain switched off, the only possible tests are the ones done in No PoL Mode.

Thus, the following table shows the tests that are going to be carried out in Situation B.

| | DISCHARGE MODE | P8 CHARGE MODE | P1-P6 CHARGE MODE |
|---|---|---|---|
| No PoL | | X | |
| Released Mode | | | |
| Pre-Detumbling Mode | | | |
| Detumbling Mode | | | |
| Detumbled (Sun - Safe) Mode | | | |
| Survival Mode | | | |
| Nominal Mode | | | |

## D.1- No PoL Mode

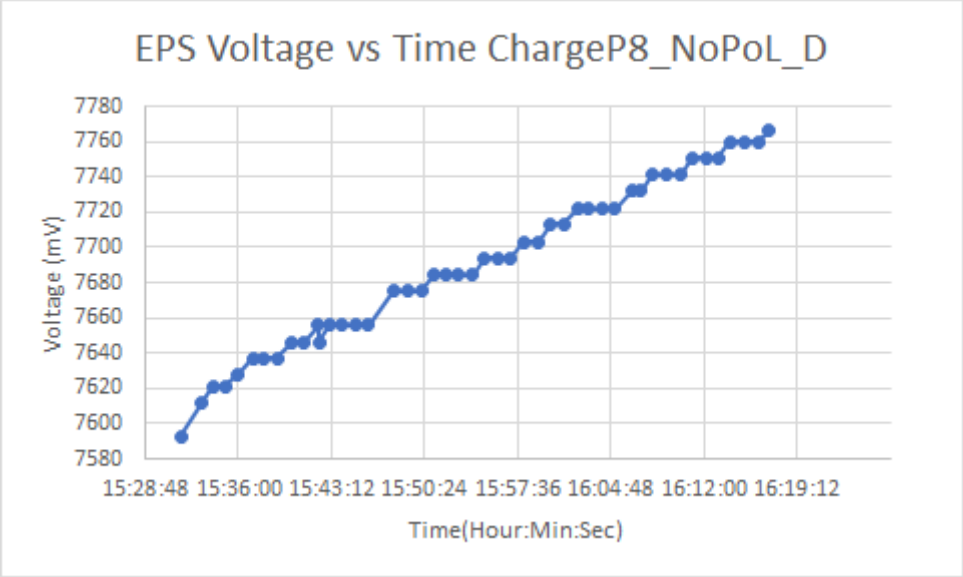
We have put the No PoL Mode, but as the EPS is disconnected, it actually does not matter whether the subsystems are connected or not. The only tests that make sense in this case are the P8 Charge Mode and the P1-P6 Charge Mode, but the mission modes can not be studied.

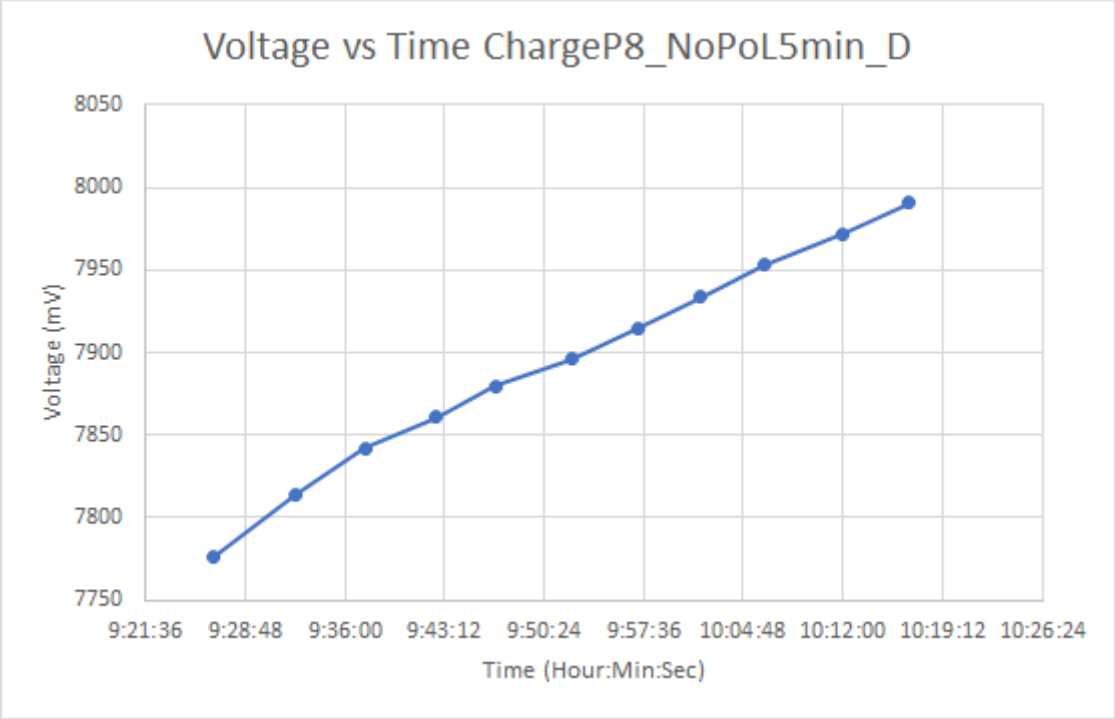
### D.1.1- Discharge Mode

As the EPS is actually disconnected, there can be no discharge mode.

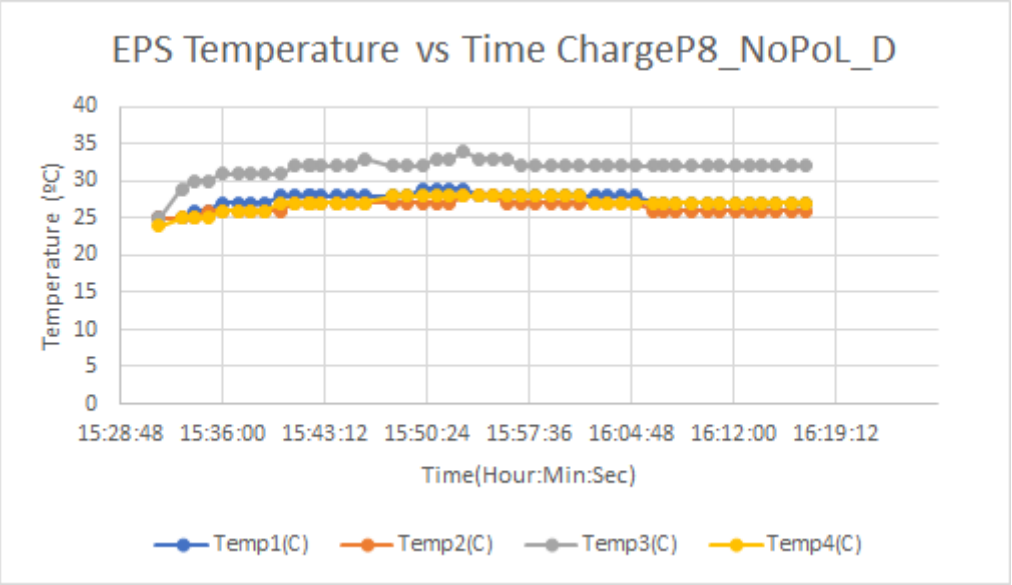
### D.1.2- P8 Charge Mode

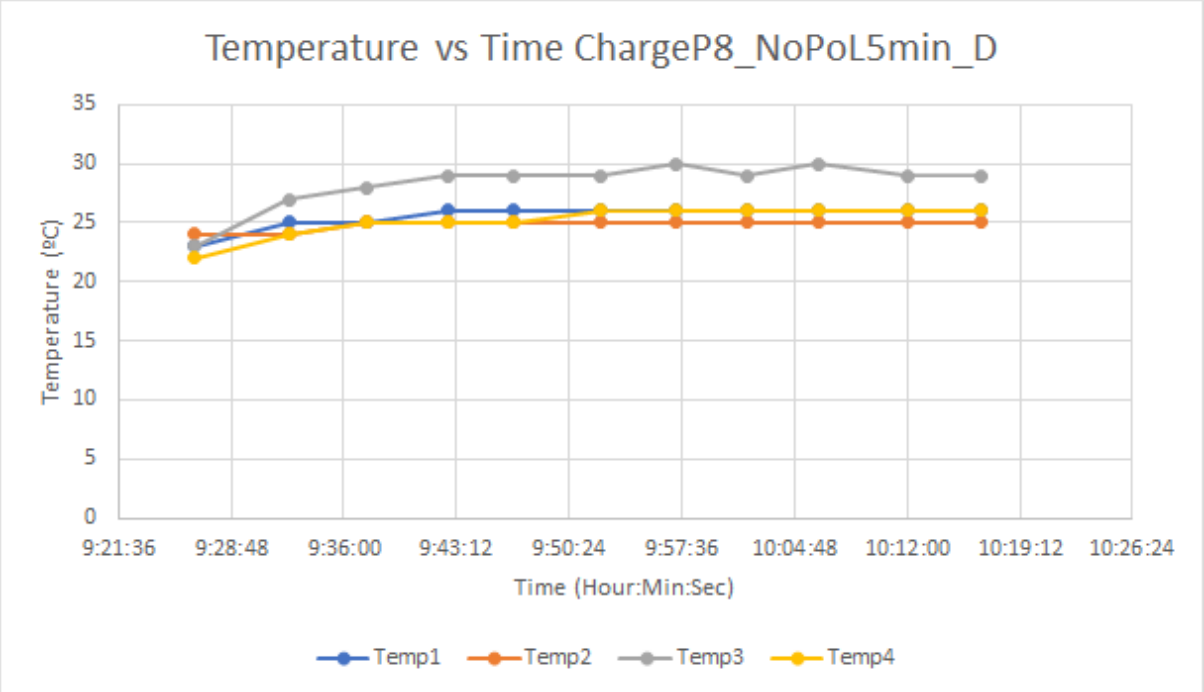
- **Voltage plot.**



- **Temperature plot.**




Despite being the first charge mode that actually charges considerably the EPS, the Temperature values are not completely right. The T3 Temperature Sensor (which is the hottest sensor) shows a Temperature incorrect, because it was way hotter when we were taking out the test.

### D.1.3- P1-P6 Charge Mode

We get the same observation as mentioned in B.1.3.

Thus, as the results obtained in the B.1.3 do not show a charge configuration (in charge mode, the EPS is oscillating between 2 values but it is not charging) we consider this test is not necessary, because we will see the same results.

# CONCLUSIONS

The main objective of this document is to characterize the performance of the Electrical Power Subsystem in the different configurations.
As we have seen, the only charge mode that actually works is the Situation D (both EPS and Raspberry disconnected throughout the charging period) charging through the P8 pin connector. The other modes do not work probably due to the aforementioned charge effects.

Although we could charge the EPS with this method, we consider that the T3 Temperature Sensor is not trustworthy. Despite giving some sensible temperature values, we proved that the sensor was way hotter than the given Temperature value. Therefore, we understand we maybe should make a test with a Thermocouple connected to that sensor before heating the EPS.

It seems that there are no problems in the discharge mode taking into account the Temperature values obtained in each Discharge Test. In this case, we have proved that the Temperature values are similar with the sensors' Temperature.

## 8.5. EPS TSTP

NANOSAT LAB

# Test Specifications and Test Procedures Standalone Subsystem Validation

*Electrical Power Subsystem Environmental Test Campaign*

*University Name:* Universitat Politècnica de Catalunya (UPC)
Campus Nord, building D3
08034 Barcelona, SPAIN
Date: 02/02/2020
*Reference:* 3C4_TSTP_EPS_v1.0
*Distribution List:*

| Destination | Contact |
|---|---|
| [3]Cat-4 Team | 3cat4@tsc.upc.edu |

## Authors

| First Name | Last Name | Contact | Role |
|---|---|---|---|
| Juanjo | Medina Musellas | juanjomedinamusellas@gmail.com | --- |
| Albert | Rodríguez Casellas | albert.rodriguez.casellas@gmail.com | --- |

## Revised by

| First Name | Last Name | Contact | Role |
|---|---|---|---|
| Lara | Fernandez Capon | lara-pilar.fernandez@tsc.upc.edu | System Engineer |

## Change Log

| Revision | Release date | Section | Description |
|---|---|---|---|
| | | | |

# Applicable Documents

| AD# | Reference | Revision |
|---|---|---|
| [AD1] | 3Cat4 TS-VCD | 1.6 |

# List of Abbreviations

| | |
|---|---|
| EGSE | Electrical Ground Support Equipment |
| EPS | Electrical Power System |
| ETC | Environment Test Campaign |
| FFT | Full Functional Test |
| MGSE | Mechanical Ground Support Equipment |
| MTQ | Magnetorquer |
| PoL | Point of Load |
| RBF | Remove Before Flight |
| SS | Standalone Subsystem |
| SSV | Standalone Subsystem Verification |
| TRP | Temperature Reference Point |
| TVAC | Thermal Vacuum Chamber |

# Table of contents

# List of Figures

# List of Tables

# Document Scope

This document aims to explain which are the specifications and procedures to verify the correct functionality of the Electrical Power Subsystem (EPS) under space conditions.

An Acceptance Test is going to be carried out to test the different parameters under study. Therefore, the test will be done and repeated 4 times with a temperature range that will be reduced by a 10 degrees margin for the acceptance test of the subsystem and a range 5 degrees wider than the design temperature range, as explained afterwards.

As a matter of fact, the parameters studied throughout the process are going to be examined in their own test. They can be divided into:

- Functional test: Its purpose will be to get the EPS housekeeping, such as the system current, the batteries voltage and the temperature obtained by the EPS temperature sensors.
- Batteries charge and discharge test: Within the Acceptance test, there will be two plateaus (stable zones to collect information and interact with the subsystem) in which the batteries will be discharging and the other two where the batteries will have a charge behavior.

This document has been written with the intention of guaranteeing the correct execution of the tests in the UPC NanoSat Lab facilities. For this reason, the test schedules have been made based on these facilities conditions.

Every test description contains the verification requirements needed to determine if the test results are satisfactory and if the system has passed the tests. It also contains the test requirements that define the conditions on which the test should be carried out.

Regarding the set-up facilities, every test description includes a detailed list about the Ground Support Equipment (MEGSE, EGSE) and the necessary tools to perform each test. Moreover, it is going to be included a step-by-step procedure to succeed in the execution of every test.

# Environment TSTP

## Thermal TSTP

### EPS Subsystem Acceptance

#### *Test Description and Objectives*

The test consists of verifying the EPS behavior with the cold and hot operational temperature (both in charge and discharge mode) and verifying the correct functionality of the system.

As it is already mentioned, the type of test to carry out is an Acceptance test, which states that there are going to be 4 cycles. The measures will be taken in every plateau and with +/- 10ºC of margin with respect to the operational temperature. Nevertheless, the last two cycles will be an Acceptance Test for our specific mission, not for the operational temperature of the subsystem.

In order to do that, the design temperature is established with a +/- 10ºC margin from the operational temperature. The mission temperatures must be inside the design range.

Once we have checked this, the acceptance test will be carried out at +/- 5ºC, and if it is a qualification test, at +/- 10ºC, always in the worst scenario.

Each state has its own maximum temperatures, so the test is designed not to exceed that limits but to stay very close.

Moreover, there are different purposes to be accomplished, which will be separated in the study process:

- **Functional test**

  The functional test provides some information about the behavior of the system throughout the test.

  In order to be able to study such behavior, it has been developed a script which gathers data from the system. The communication with the On-Board Computer is possible thanks to the connection of the subsystems by the I2C protocol.

Concretely, the parameters put into the study will be the information obtained by the 4 temperature sensors in the EPS, the state of every Point of Load and the system's current in each state.

As there is no information about the Voltage variation with the Temperature, the temperature and voltage information gathered in this process will be used to perform a **characterization test**, which is going to reflect the Tension variation with the Temperature.

- **Batteries charge and discharge test**

  As the whole Acceptance test is composed by four plateaus, two of them are going to be in the batteries charge state and the other two of them in the discharge state.

  In each plateau, every Point of Load (simulating the activation of a determined subsystem) will be activated according to the mission modes (NoPoL, Survival and Nominal) and will be evaluated if the current rise corresponds to the expected value.

The whole Acceptance test is performed under vacuum conditions, with the Engineering Model of the EPS assembled into a test bench inside the chamber.

In order to have a reference to compare with, the first test that will be done is a **Full Functional Test (FFT)**, which is a Functional Test with the TVAC chamber opened, that means it is done under room pressure and temperature. The data obtained in the test will be used to compare it with the data gathered with the other tests in the whole process.

The **Full Functional Test** will consist in the execution of the C script in order to get the housekeeping and check if the PoLs can be enabled and disabled. Moreover, it must be verified that the EPS can be charged. The FFT will pass if every part described before is successfully carried out.

The test requires a pressure level lower than 10E-5 mbar and the thermal profile is as shown in Figure 1.

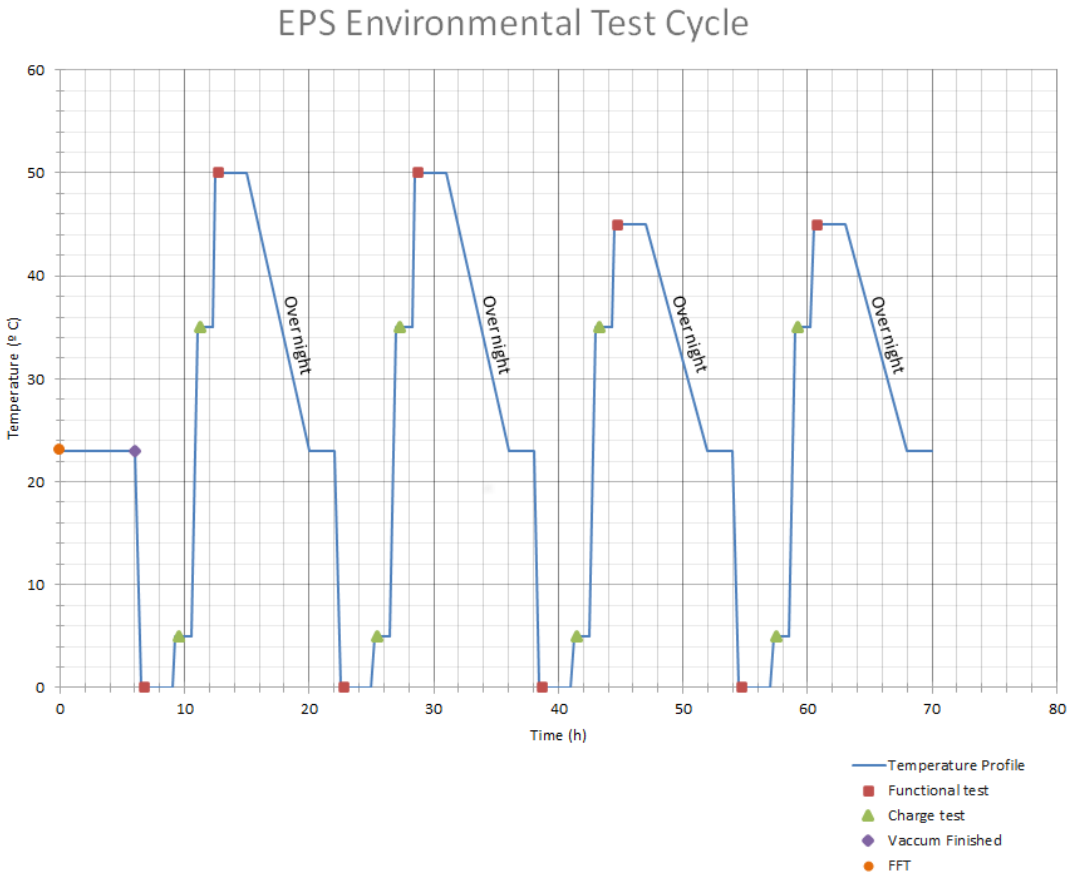



**Figure 1 - Thermal profile of EPS Subsystem Acceptance**

### *Requirements Verification*

In order to perform properly, the EPS should fulfil the following requirements*:

The EPS shall have the capability to provide the following housekeeping data:

- Battery Voltage / State of Charge
- Total spacecraft current consumption
- 5.0 V DC bus voltage monitoring
- Current consumption of the 5.0 V DC bus switched outputs
- Overcurrent events in 5.0 V DC bus switched outputs
- 3.3 V DC bus voltage monitoring
- Current consumption of the 3.3 V DC bus switched outputs
- Overcurrent events in 3.3 V DC bus switched outputs

- Temperature readings

The EPS shall forward all housekeeping data to the OBC

The EPS shall execute specific commands from the OBC in order to turn on and off the different power outputs

The EPS battery operating temperature range shall be from 3ºC to 40 ºC.

**Data retrieved from 3Cat4 TS-VCDv1.6 [AD1]*

### *Test Requirements*

The test requires a Thermal Vacuum Chamber (TVAC), the subsystem shall be placed inside the chamber.

### *Test Organization and Schedule*

The tests are performed by two operators. The general outline of the Tests can be seen in Table 5.

As it is going to be the first time the operators carry out such kind of test, both of them are going to perform the same tasks so as to acquire the necessary supervision.

| **Test Phase** | Time required |
|---|---|
| Integration | 120 minutes |
| Full Functional Test | 210 minutes |
| Vacuum | 72 hours |
| Temperature ramp and thermal stabilisation (0ºC) | 45 minutes |
| Plateau (Discharge) Functional Test | 120 minutes |
| Temperature ramp and thermal stabilisation (5ºC) | 30 minutes |
| Plateau (Charge) Functional Test | 45 minutes |
| Temperature ramp and thermal stabilisation (35ºC) | 45 minutes |
| Plateau (Charge) Functional Test | 45 minutes |
| Temperature ramp and thermal stabilisation (50ºC) | 30 minutes |
| Plateau (Discharge) Functional Test | 120 minutes |
| Temperature ramp to ambient temperature | 5 hours |
| Pressurization | 72 hours |

**Table 1 - Test outline**

### *Test Setup*

The setup must start shorting both pins in each P10 and P11 connectors. This will disable kill switches and allow the EPS to turn on.

The connections between the simulated subsystems and the EPS (each one in its PoL) are shown in the following table and the perfboard with the resistors is shown after it:

| Subsystems | PoL | Pin | Ground | Current (A) | Computed Resistors (ohms) | Actual Resistors (ohms) |
|---|---|---|---|---|---|---|
| OBC Mean Current(3.3V) | 4 | H1-48 | H2-29 | 0,052 | 64 | 62 |
| UHF/AIS and Helix Mean Current (3.3V) | 5 | H1-50 | H2-29 | 0,016 | 205 | 200 |
| ADCS Mean Current (3.3V) | 6 | H1-52 | H2-29 | 0,091 | 36 | 38 |
| COMMS Mean Current(5V) | 1 | H1-47 | H2-29 | 0,051 | 98 | 100 |
| MTQ Mean Current (5V) | 2 | H1-51 | H2-29 | 0,198 | 76* | 75* |
| Payload Mean Current(5V) | 3 | H1-49 | H2-29 | 0,065 | 77 | 75 |
| TOTAL SUM | | | | 0,473 | | |

**Table 2 - PoL - subsystems connections**

*In this subsystem we are going to put 3 resistors in parallel of 75 ohms because there are 3 MTQ, that is because the current is that high.

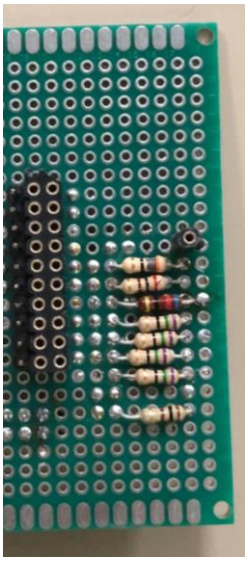
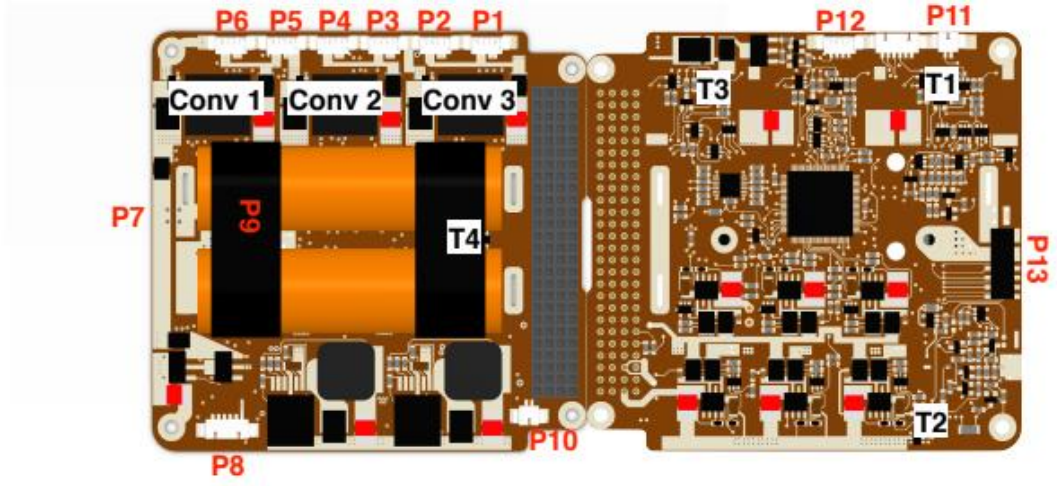
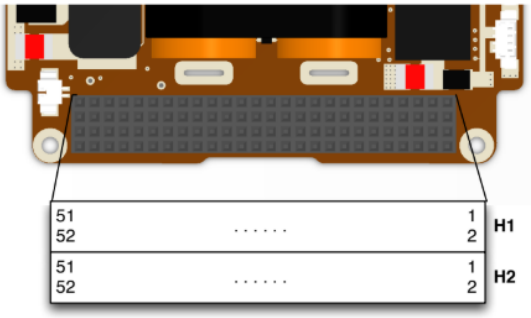


**Figure 2 - Resistors perfboard, simulating every subsystem**

As it can be seen in the aforementioned perfboard picture, the resistors are sorted from 1 to 6 from the ground up.




*Stack connector pinout*

| P8<br>Flight Preparation Panel connector | |
|---|---|
| Pin | Usage |
| 1 | GND |
| 2 | RBF[1] |
| 3 | KS[2] |
| 4 | KS_RESET[3] |
| 5 | Vbat (through 100kOhm)[4] |
| 6 | CHARGE |

| P1 to P6<br>Solar panel input connectors | |
|---|---|
| Pin | Usage |
| 1 | GND |
| 2 | GND |
| 3 | Vsc |
| 4 | Vsc |

| P10 and P11<br>Kill Switch Connectors | |
|---|---|
| Pin | Usage |
| 1 | KS- (GND)[5] |
| 2 | KS+[5] |

| P12<br>Serial connector | |
|---|---|
| Pin | Usage |
| 1 | GND |
| 2 | Not connected |
| 3 | RxD |
| 4 | TxD |

**Figure 3 - EPS thermal sensors and connectors**

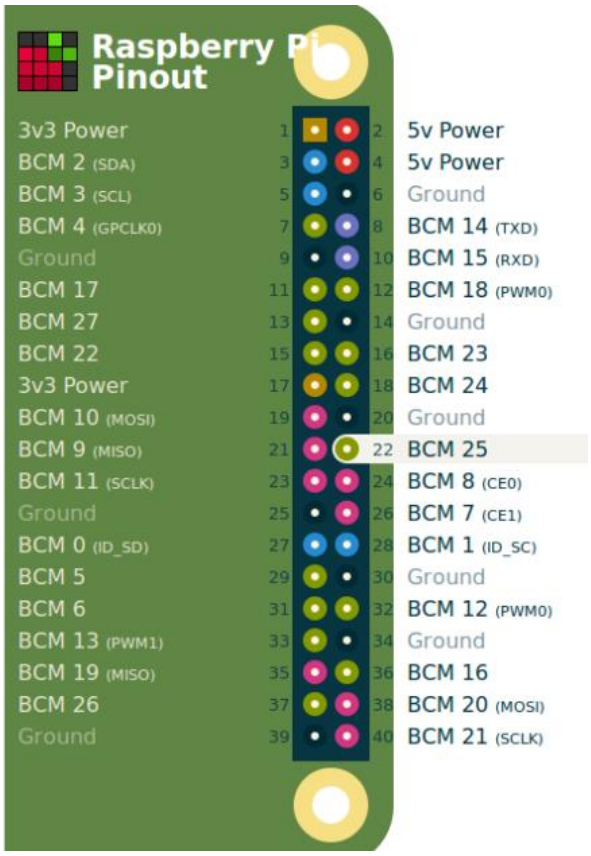



**Figure 4 - OBC (Rapberry) pinout**

## Item Under Test

| Identifier | Description | Item | Nº | Figure |
|---|---|---|---|---|
| IUT_1 | EPS EM | Electrical Power Subsystem Engineering Model | 1 |  |
| IUT_2 | Resistors that simulate the different subsystems that would be connected to the EPS by the PoL. 3 Resistors will be in the PoL of 3.3V and the other 3 will be in the 5 V PoL. | Resistors (simulated subsystems) connected to the EPS (see Table 2 for the details). | 6 |  |

**Table 3 - Items under test for the Test Set-up**

## **MGSE** (Mechanical Ground Support Equipment)

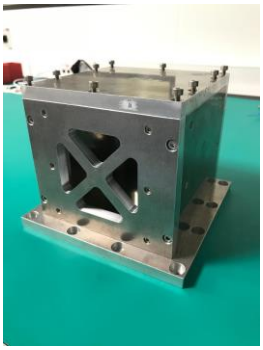
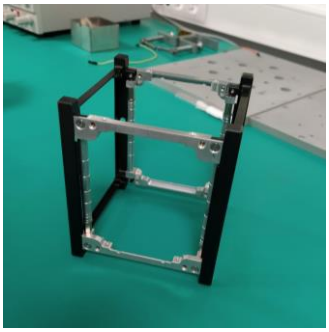
| Identifier | Description | Item | Nº | Figure |
|---|---|---|---|---|
| MGSE_1 | Aluminium Box | Chamber to distribute uniformly the temperature change | 1 |  |
| MGSE_2 | CubeSat Structure | CubeSat structure to assemble the EPS within it | 1 |  |

**Table 4 - Mechanical Equipment for the Test Set-up**

## **EGSE** (Electrical Ground Support Equipment)

| ID | Description | Item | Nº | Figure |
|---|---|---|---|---|
| EGSE_1 | Power Supply | Minimum 1A at 5V. One channel | 1 |  |
| EGSE_2 | Micro USB plug | Used to feed the OBC Simulator | 1 |  |
| EGSE_3 | OBC Simulator | Raspberry Pi | 1 |  |
| EGSE_4 | External harness | Wire that connects the external part of the DSub-9 feedthrough to the OBC Simulator and the Power Supply | 1 |  |
| EGSE_5 | Internal harness | Wire that connects the internal part of the DSub-9 feedthrough to the EPS mainboard | 1 |  |
| EGSE_6 | Router or switch | Router or switch with power cable and Ethernet cable to create a WLAN to communicate with the OBC simulator | 1 |  |

**Table 5 – Electrical Ground Support Equipment for the Test Set-up**

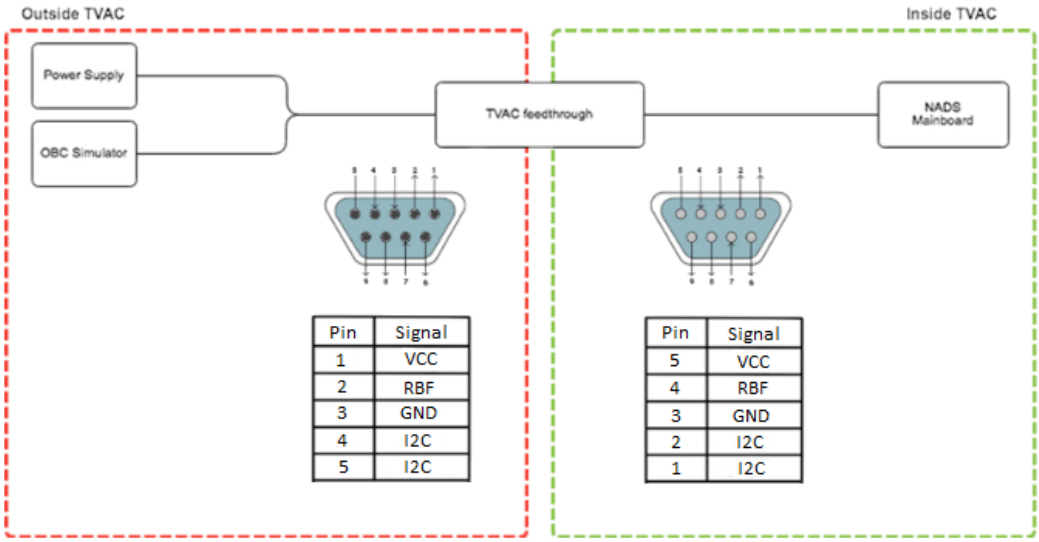



**Figure 5 - Pinout for the Subsystem Acceptance Test**

## Tools

| ID | Description | Item | Nº | Figure |
|---|---|---|---|---|
| T_1 | Laptop | To follow the procedures | 1 |  |
| T_2 | Watch | Minutes precision | 1 |  |
| T_3 | Screwdrivers | TX 8, TX 9 | 1 |  |

| | | | | |
|---|---|---|---|---|
| T_4 | Wrench Keys | Allen set | 1 |  |
| T_5 | Nut Driver | SW 5.5 mm | 1 |  |
| T_6 | Torque Screwdriver 0.5 to 2.5 Nm | TX 8, TX0, Allen | 1 |  |
| T_7 | Pliers | Mini Pliers Set | 1 |  |
| T_8 | Cutting tool | Scissors, Cutter | 1 |  |
| T_9 | Kapton tape | 5 mm | 1 |  |
| T_10 | Tweezers | Tweezer set | 1 |  |

**Table 6 - Tools for the Test Set-up**

### *Tolerance Range or Pass/Fail Criteria*

There are different criteria depending on the test we are referring to. As there are two tests to perform, two different criteria are defined:

- **Functional test:**

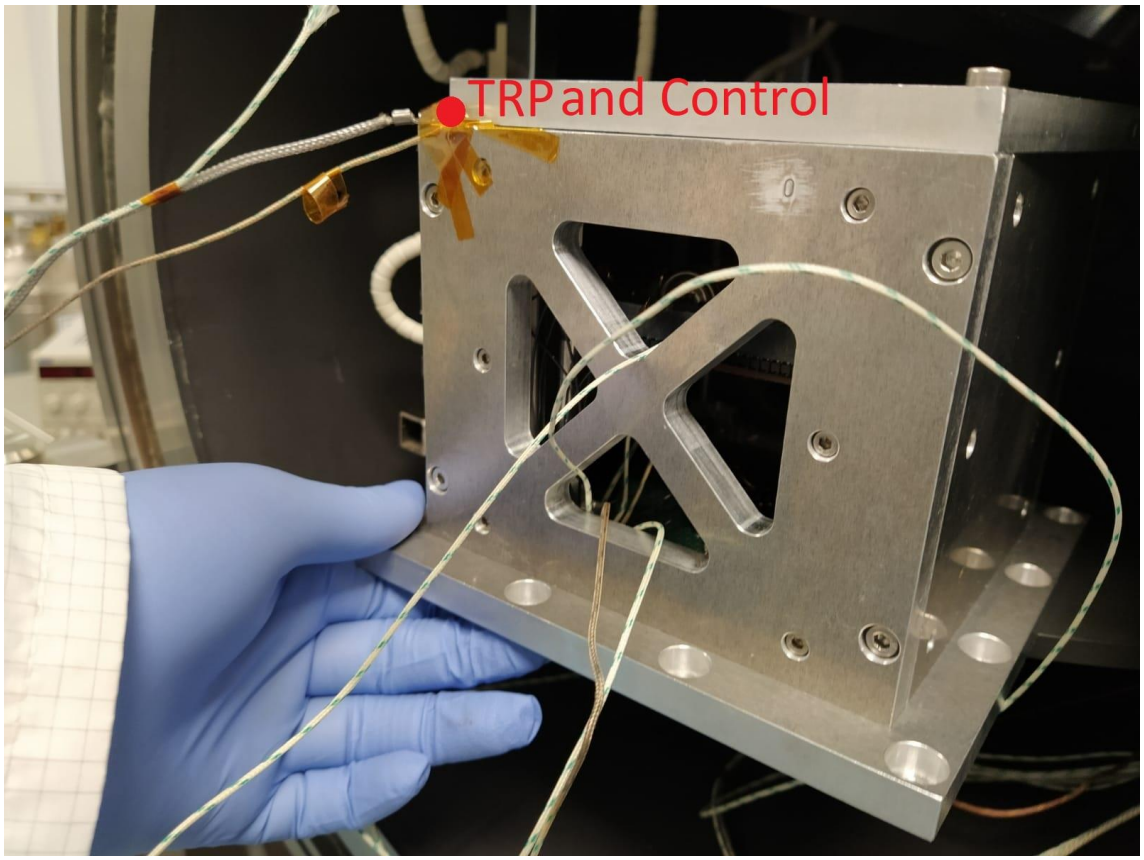
This test will consist in sending a script to gather the subsystem housekeeping. Therefore, the test will perform properly as long as the EPS returns all the information that is required, which is: battery voltage, sensors temperature and number of PoL activated and deactivated. Also, as it has to be correct, it will be compared with the thermocouples in order to have a reference measurement and the batteries' voltage range stated in the batteries' datasheet, which means a minimum voltage of 3 V and a maximum voltage of 4.2 V for each battery, so as an overall, the total minimum value will be 6 V and the maximum 8.4 V.

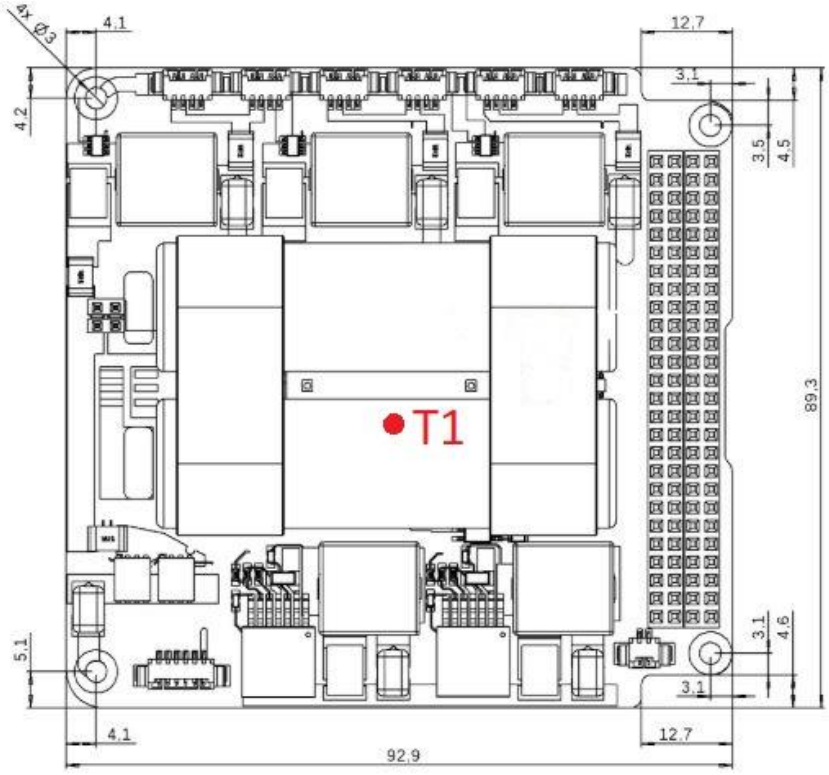
- **Batteries charge and discharge test:**

The aim of this test is to verify the correct functionality of the subsystem in charge and discharge. Thus, it will be considered as a successful test if the functional and characterization test finish successfully in both states (charge and discharge).

### *Thermocouples placing*

There will be a thermocouple close to each temperature sensor and one in the processor. Thus, there would be 5 thermocouples controlling the performance of the EPS temperature sensors, processor and batteries, but as it must be a Control thermocouple (the one which does a redundancy check in the Aluminium Box temperature) and the laboratory only has 5 thermocouples, one of the temperature sensors will be dismissed. The Temperature Reference Point will be placed in the MGSE_1, which is the Aluminium Box with the Control.


TRP and Control


4x Ø3
4.1
12.7
3.1
4.2
3.5
4.5
T1
89.3
5.1
3.1
4.6
3.1
4.1
12.7
92.9

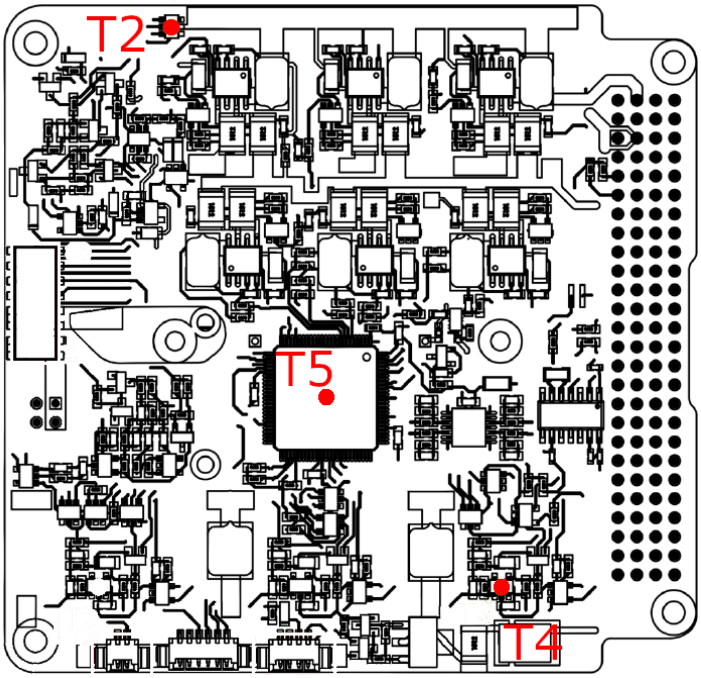



**Figure 6 - Location of the thermocouples in the EPS**

### *Step-by-Step Procedure*

**TSTP-SSV-EPS-001: EPS Acceptance Test**

**Date: 15/07/2020**

**Location: Clean room with TVAC**

**Activity description: Test to verify the functionality of the EPS under acceptance environment conditions**

**Operators: Albert Rodríguez, Juanjo Medina**

| Step ID | Instruction | Expected | Actual | Pass/Fail Criteria | Passed [Y/N] | Time | PA |
|---|---|---|---|---|---|---|---|
| ETC-EPS-001-010 | Assemble the EPS to the CubeSat structure | EPS assembled to the structure | | EPS firmly assembled | | | |
| ETC-EPS-001-020 | Connect the resistors to the required EPS PoL (connections properly specified in section 1.1.1.5) | Each resistor properly connected | | Connections firmly plugged | | | |
| ETC-EPS-001-030 | Encapsulate the EPS into the Aluminium Box | EPS encapsulated into the Aluminum Box | | EPS firmly encapsulated into the Aluminium Box | | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **ETC-EPS-001-040** | **Connect the cables from the DSub-9 to the EPS Mainboard (connect the inside DSub-9 Pin5 (CHARGE), Pin4 (RBF), Pin3 (GND), Pin2 (SCL) and Pin1 (SDA) to the EPS P8 Pin6 (CHARGE), the P8 Pin2 (RBF), the P8 Pin1 (GND), the H1-43 (SCL) and the H1-41 (SDA) respectively. Pass the cables through the Aluminium Box aperture for wiring. Connect the EPS P7 jumper to connect the batteries to the EPS circuit.** | **Mainboard connected through cable to DSub-9.** | | **All connections firmly plugged.** | | | |
| **ETC-EPS-001-050** | **Place thermocouples 1, 2, 4, 5 Control and TRP on the EPS using kapton. Location specified in section 1.1.1.7.** | **Thermocouples in place** | | **Thermocouples fixed in designed places.** | | | |
| **ETC-EPS-001-060** | **Place the EPS into the TVAC** | **EPS placed into the TVAC** | | **EPS firmly placed into the TVAC** | | | |
| **ETC-EPS-001-070** | **Connect the DSub-9 Pin3 (GND) to the Raspberry Pin 39 (GND), the DSub-9 Pin4 (SCL) to the Pin 3 (SCL) of the Raspberry and the DSub-9 Pin5 (SDA) to the Pin 2 of the Raspberry (SDA). Connect the supply wire of the OBC Simulator (Raspberry). Connect the OBC Simulator with the computer by an Ethernet wire. Switch on the Power Supply and configure it with a 5V and 1A limit, but don't connect it to the DSub-9 yet. Connect the DSub-9 Pin2 (RBF) to the DSub-9 Pin3 (GND) to keep the EPS switched off.** | **OBC Simulator, Power Supply and EPS are turned on, and the Power Supply is properly configured.** | | **All connections firmly plugged and each device switched on. Also, we can see 5V and a 1A limit in the Power Supply screen.** | | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **ETC-EPS-001-080** | **Start of the FFT in the EPS. Disconnect the DSub-9 Pin 2 (RBF) from the DSub-9 Pin3 (GND) in order to switch on the EPS in discharge mode. As it is mentioned previously, once the RBF and GND are disconnected, the EPS is switched on. Connect the computer with the OBC Simulator through SSH protocol (type ssh pi@rasp_ip_address). (Raspberry password: raspberry). You can know the Raspberry IP address by typing the command "nmap 10.4.33.0-255" .** | **Connection between the computer and the OBC and EPS switched on** | | **We are able to send a ping from the computer to the assigned IP direction of the Raspberry.** | | | |
| **ETC-EPS-001-090** | **Execution of the C script (./I2C). Once done, type "get" in the command prompt to extract the EPS housekeeping** | **Extraction of the EPS housekeeping information and visualization of it by the command prompt** | | **All the data (PoL, vbatt, Temperatures,etc. ) is correctly seen in the command prompt** | | | |
| **ETC-EPS-001-100** | **Type the command "register Discharge_NoPoL_Functional.csv NoPoL" and wait 15 min.** | **Extraction of the EPS housekeeping information and visualization of it by the command prompt during 15 min without any load connected** | | **All the data (PoL, vbatt, Temperatures,etc. ) is correctly seen in the command prompt.Current values match with values in Table 2 and temperatures match with the temperatures shown by the TVAC** | | | |
| **ETC-EPS-001-110** | **Type the command "register Discharge_Nominal_Functional.csv Nominal" and wait 45 min.** | **Extraction of the EPS housekeeping information and visualization of it by the command prompt during 45 min in Nominal Mode (each PoL connected except PoL 2 that is only the first 9 min)** | | **All the data (PoL, vbatt, Temperatures,etc. ) is correctly seen in the command prompt.Current values match with values in Table 2 and temperatures match with the temperatures shown by the TVAC** | | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **ETC-EPS-001-120** | **Type the command "register Discharge_Survival_Functional.csv Survival" and wait 45 min.** | **Extraction of the EPS housekeeping information and visualization of it by the command prompt during 45 min in Survival Mode (PoL 1,4,6 active and also 2 the first 9 min)** | | **All the data (PoL, vbatt, Temperatures,etc. ) is correctly seen in the command prompt.Current values match with values in Table 2 and temperatures match with the temperatures shown by the TVAC** | | | |
| **ETC-EPS-001-130** | **Start of the Charge test. Go to the C script (nano I2C.c) and change the value of the variable called "filename" in the file_log function to "Charge_Functional.csv". Then, type the command "gcc I2C.c -o I2C" to compile the script.** | **C script compiled successfully** | | **There is no error message when typing the compiling command** | | | |
| **ETC-EPS-001-140** | **Switch off the EPS (connect the RBF to GND) and disconnect the Raspberry GND to disconnect the Raspberry from the EPS (and thus eluding the charge effect). Connect the DSub-9 Pin 1 (Vcc) with the positive cable of the Power Supply and the D-Sub9 Pin 3 (GND) with the Power Supply's ground. Every 5 min, disconnect the Power Supply wires, and then connect again the Raspberry GND and disconnect the RBF from the GND to switch ON the EPS. Execute the script (./I2C) and type "get" and then "log". and then disconnect again the Raspberry and EPS and connect the Power Supply wires.Repeat the process every 5 min during 45 min. Once done that, the FFT is finished.** | **Extraction of the EPS housekeeping information in charge mode and visualization of it by the command prompt during 45 min.** | | **All the data (PoL, vbatt, Temperatures,etc. ) is correctly seen in the command prompt** | | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **ETC-EPS-001-150** | **Switch off the EPS by connecting the DSub-9 Pin2 (RBF) to the DSub-9 Pin3 (GND) and disconnect the GND from the Raspberry.** | **There is no communication between the EPS and the OBC.** | | **DSub-9 Pin2 (RBF) and DSub-9 Pin3 (GND) are connected. If the get command inside the I2C script is executed, there is no information extracted.** | | | |
| **ETC-EPS-001-160** | **Verify the thermocouples sense temperature** | **Thermocouples are active** | | **Thermocouples reading an expected value (room temperature)** | | | |
| **ETC-EPS-001-170** | **Close TVAC.** | **TVAC closed** | | **You cannot open TVAC** | | | |
| **ETC-EPS-001-180** | **Proceed to vacuum TVAC until 10E-5 mbar are reached.** | **Power TVAC.** | | **10E-5 mbar are reached.** | | | |
| **ETC-EPS-001-190** | **Cool TVAC environment and SS to 0ºC** | **Cooling of the SS.** | | **0ºC reached by the SS.** | | | |
| **ETC-EPS-001-200** | **Wait 15 min at 0ºC (dwell).** | **No temperature fluctuations during 15 min.** | | **No temperature fluctuations during 15 min.** | | | |
| **ETC-EPS-001-210** | **Start of the Functional Test in the EPS. Disconnect the DSub-9 Pin2 (RBF) from the DSub-9 Pin3 (GND) in order to switch on the EPS in discharge mode and connect the GND to the Raspberry.Connect the computer with the OBC Simulator through SSH protocol.** | **Connection between the computer and the OBC and EPS switched on** | | **We are able to send a ping from the computer to the assigned IP direction of the Raspberry.** | | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **ETC-EPS-001-220** | **Execution of the C script (./I2C). Once done, type "get" in the command prompt to extract the EPS housekeeping** | **Extraction of the EPS housekeeping information and visualization of it by the command prompt** | | **All the data (PoL, vbatt, Temperatures,etc. ) is correctly seen in the command prompt.Current values match with values in Table 2 and temperatures match with the temperatures shown by the TVAC** | | | |
| **ETC-EPS-001-230** | **Type the command "register NoPoL_T0_C1.csv NoPoL" and wait 15 min.** | **Extraction of the EPS housekeeping information and visualization of it by the command prompt during 15 min without any load connected** | | **All the data (PoL, vbatt, Temperatures,etc. ) is correctly seen in the command prompt.Current values match with values in Table 2 and temperatures match with the temperatures shown by the TVAC** | | | |
| **ETC-EPS-001-240** | **Type the command "register Survival_T0_C1.csv Survival" and wait 45 min.** | **Extraction of the EPS housekeeping information and visualization of it by the command prompt during 45 min in Survival Mode (PoL 1,4,6 active and also 2 the first 9 min)** | | **All the data (PoL, vbatt, Temperatures,etc. ) is correctly seen in the command prompt.Current values match with values in Table 2 and temperatures match with the temperatures shown by the TVAC** | | | |
| **ETC-EPS-001-250** | **Type the command "register Nominal_T0_C1.csv Nominal" and wait 45 min.** | **Extraction of the EPS housekeeping information and visualization of it by the command prompt during 45 min in Nominal Mode (each PoL connected except PoL 2 that is only the first 9 min)** | | **All the data (PoL, vbatt, Temperatures,etc. ) is correctly seen in the command prompt.Current values match with values in Table 2 and temperatures match with the temperatures shown by the TVAC** | | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **ETC-EPS-001-260** | **End of the Functional Test. Switch off the EPS by connecting the DSub-9 Pin2 (RBF) to the DSub-9 Pin3 (GND) and disconnect the DSub-9 Pin3 (GND) from the Raspberry.** | **There is no communication between the EPS and the OBC.** | | **DSub-9 Pin2 (RBF) and DSub-9 Pin3 (GND) are connected. If the get command inside the I2C script is executed, there is no information extracted.** | | | |
| **ETC-EPS-001-270** | **Heat TVAC environment and SS to 5ºC.** | **Heating of the SS.** | | **5ºC reached by the SS.** | | | |
| **ETC-EPS-001-280** | **Wait 15 min at 5ºC (dwell).** | **No temperature fluctuations during 15 min.** | | **No temperature fluctuations during 15 min.** | | | |
| **ETC-EPS-001-290** | **Start of the Functional Test in charge mode. Go to the C script (nano I2C.c) and change the value of the variable called "filename" in the file_log function to "Charge_T5_C1.csv". Then, type the command "gcc I2C.c -o I2C" to compile the script.** | **C script compiled successfully** | | **There is no error message when typing the compiling command** | | | |
| **ETC-EPS-001-300** | **Connect the D-Sub9 Pin 1 (Vcc) with the positive cable of the Power Supply and the D-Sub9 Pin 3 (GND) with the Power Supply's ground. Every 5 min, disconnect the Power Supply wires, and then connect again the Raspberry GND and disconnect the RBF from the GND to switch ON the EPS. Execute the script (./I2C) and type "get" and then "log". and then disconnect again the Raspberry and EPS and connect the Power Supply wires.Repeat the process every 5 min during 45 min. Once done that, the test is finished.** | **Extraction of the EPS housekeeping information in charge mode and visualization of it by the command prompt during 45 min.** | | **All the data (PoL, vbatt, Temperatures,etc. ) is correctly seen in the command prompt** | | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **ETC-EPS-001-310** | **Switch off the EPS by connecting the D-Sub9 Pin2 (RBF) to the DSub-9 Pin3 (GND) and disconnect the GND from the Raspberry.** | **There is no communication between the EPS and the OBC.** | | **DSub-9 Pin2 (RBF) and DSub-9 Pin3 (GND) are connected. If the get command inside the I2C script is executed, there is no information extracted.** | | | |
| **ETC-EPS-001-320** | **Heat TVAC environment and SS to 35ºC.** | **Heating of the SS.** | | **35ºC reached by the SS.** | | | |
| **ETC-EPS-001-330** | **Wait 15 min at 35ºC (dwell).** | **No temperature fluctuations during 15 min.** | | **No temperature fluctuations during 15 min.** | | | |
| **ETC-EPS-001-340** | **Start of the Functional Test in charge mode. Go to the C script (nano I2C.c) and change the value of the variable called "filename" in the file_log function to "Charge_T35_C1.csv". Then, type the command "gcc I2C.c -o I2C" to compile the script.** | **C script compiled successfully** | | **There is no error message when typing the compiling command** | | | |
| **ETC-EPS-001-350** | **Connect the D-Sub9 Pin 1 (Vcc) with the positive cable of the Power Supply and the D-Sub9 Pin 3 (GND) with the Power Supply's ground. Every 5 min, disconnect the Power Supply wires, and then connect again the Raspberry GND and disconnect the RBF from the GND to switch ON the EPS. Execute the script (./I2C) and type "get" and then "log". and then disconnect again the Raspberry and EPS and connect the Power Supply wires.Repeat the process every 5 min during 45 min. Once done that, the test is finished.** | **Extraction of the EPS housekeeping information in charge mode and visualization of it by the command prompt during 45 min.** | | **All the data (PoL, vbatt, Temperatures,etc. ) is correctly seen in the command prompt** | | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **ETC-EPS-001-360** | **Switch off the EPS by connecting the DSub-9 Pin2 (RBF) to the DSub-9 Pin3 (GND) and disconnect the GND from the Raspberry.** | **There is no communication between the EPS and the OBC.** | | **DSub-9 Pin2 (RBF) and DSub-9 Pin3 (GND) are connected. If the get command inside the I2C script is executed, there is no information extracted.** | | | |
| **ETC-EPS-001-370** | **Heat TVAC environment and SS to 50ºC.** | **Heating of the SS.** | | **50ºC reached by the SS.** | | | |
| **ETC-EPS-001-380** | **Wait 15 min at 50ºC (dwell).** | **No temperature fluctuations during 15 min.** | | **No temperature fluctuations during 15 min.** | | | |
| **ETC-EPS-001-390** | **Start of the Functional Test in the EPS. Disconnect the DSub-9 Pin 2 (RBF) from the DSub-9 Pin3 (GND) in order to switch on the EPS in discharge mode and connect the GND to the Raspberry.Connect the computer with the OBC Simulator through SSH protocol.** | **Connection between the computer and the OBC and EPS switched on** | | **We are able to send a ping from the computer to the assigned IP direction of the Raspberry** | | | |
| **ETC-EPS-001-400** | **Execution of the C script (./I2C). Once done, type "get" in the command prompt to extract the EPS housekeeping** | **Extraction of the EPS housekeeping information and visualization of it by the command prompt** | | **All the data (PoL, vbatt, Temperatures,etc. ) is correctly seen in the command prompt.Current values match with values in Table 2 and temperatures match with the temperatures shown by the TVAC** | | | |
| **ETC-EPS-001-410** | **Type the command "register NoPoL_T50_C1.csv NoPoL" and wait 15 min.** | **Extraction of the EPS housekeeping information and visualization of it by the command prompt during 15 min without any load connected** | | **All the data (PoL, vbatt, Temperatures,etc. ) is correctly seen in the command prompt.Current values match with values in Table 2 and temperatures match with the temperatures shown by the TVAC** | | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **ETC-EPS-001-420** | **Type the command "register Survival_T50_C1.csv Survival" and wait 45 min.** | **Extraction of the EPS housekeeping information and visualization of it by the command prompt during 45 min in Survival Mode (PoL 1,4,6 active and also 2 the first 9 min)** | | **All the data (PoL, vbatt, Temperatures,etc. ) is correctly seen in the command prompt.Current values match with values in Table 2 and temperatures match with the temperatures shown by the TVAC** | | | |
| **ETC-EPS-001-430** | **Type the command "register Nominal_T50_C1.csv Nominal" and wait 45 min.** | **Extraction of the EPS housekeeping information and visualization of it by the command prompt during 45 min in Nominal Mode (each PoL connected except PoL 2 that is only the first 9 min)** | | **All the data (PoL, vbatt, Temperatures,etc. ) is correctly seen in the command prompt.Current values match with values in Table 2 and temperatures match with the temperatures shown by the TVAC** | | | |
| **ETC-EPS-001-440** | **End of the Functional Test. Switch off the EPS by connecting the DSub-9 Pin2 (RBF) to the DSub-9 Pin3 (GND) and disconnect the DSub-9 Pin3 (GND) from the Raspberry.** | **There is no communication between the EPS and the OBC.** | | **DSub-9 Pin2 (RBF) and DSub-9 Pin3 (GND) are connected. If the get command inside the I2C script is executed, there is no information extracted.** | | | |
| **ETC-EPS-001-450** | **Return SS and TVAC to ambient temperature** | **Tambient is reached** | | **Temperature rises in the SS** | | | |
| **END OF CYCLE 1** | | | | | | | |
| | | | | | | | |
| **START OF CYCLE 2** | | | | | | | |
| **ETC-EPS-002-010** | **Cool TVAC environment and SS to 0ºC** | **Cooling of the SS.** | | **0ºC reached by the SS.** | | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **ETC-EPS-002-020** | **Wait 15 min at 0ºC (dwell).** | **No temperature fluctuations during 15 min.** | | **No temperature fluctuations during 15 min.** | | | |
| **ETC-EPS-002-030** | **Start of the Functional Test in the EPS. Disconnect the DSub-9 Pin2 (RBF) from the DSub-9 Pin3 (GND) in order to switch on the EPS in discharge mode and connect the GND to the Raspberry.Connect the computer with the OBC Simulator through SSH protocol.** | **Connection between the computer and the OBC and EPS switched on** | | **We are able to send a ping from the computer to the assigned IP direction of the Raspberry.** | | | |
| **ETC-EPS-002-040** | **Execution of the C script (./I2C). Once done, type "get" in the command prompt to extract the EPS housekeeping** | **Extraction of the EPS housekeeping information and visualization of it by the command prompt** | | **All the data (PoL, vbatt, Temperatures,etc. ) is correctly seen in the command prompt.Current values match with values in Table 2 and temperatures match with the temperatures shown by the TVAC** | | | |
| **ETC-EPS-002-050** | **Type the command "register NoPoL_T0_C2.csv NoPoL" and wait 15 min.** | **Extraction of the EPS housekeeping information and visualization of it by the command prompt during 15 min without any load connected** | | **All the data (PoL, vbatt, Temperatures,etc. ) is correctly seen in the command prompt.Current values match with values in Table 2 and temperatures match with the temperatures shown by the TVAC** | | | |
| **ETC-EPS-002-060** | **Type the command "register Survival_T0_C2.csv Survival" and wait 45 min.** | **Extraction of the EPS housekeeping information and visualization of it by the command prompt during 45 min in Survival Mode (PoL 1,4,6 active and also 2 the first 9 min)** | | **All the data (PoL, vbatt, Temperatures,etc. ) is correctly seen in the command prompt.Current values match with values in Table 2 and temperatures match with the temperatures shown by the TVAC** | | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **ETC-EPS-002-070** | **Type the command "register Nominal_T0_C2.csv Nominal" and wait 45 min.** | **Extraction of the EPS housekeeping information and visualization of it by the command prompt during 45 min in Nominal Mode (each PoL connected except PoL 2 that is only the first 9 min)** | | **All the data (PoL, vbatt, Temperatures,etc. ) is correctly seen in the command prompt.Current values match with values in Table 2 and temperatures match with the temperatures shown by the TVAC** | | | |
| **ETC-EPS-002-080** | **End of the Functional Test. Switch off the EPS by connecting the DSub-9 Pin2 (RBF) to the DSub-9 Pin3 (GND) and disconnect the DSub-9 Pin3 (GND) from the Raspberry.** | **There is no communication between the EPS and the OBC.** | | **DSub-9 Pin2 (RBF) and DSub-9 Pin3 (GND) are connected. If the get command inside the I2C script is executed, there is no information extracted.** | | | |
| **ETC-EPS-002-090** | **Heat TVAC environment and SS to 5ºC.** | **Heating of the SS.** | | **5ºC reached by the SS.** | | | |
| **ETC-EPS-002-100** | **Wait 15 min at 5ºC (dwell).** | **No temperature fluctuations during 15 min.** | | **No temperature fluctuations during 15 min.** | | | |
| **ETC-EPS-002-110** | **Start of the Functional Test in charge mode. Go to the C script (nano I2C.c) and change the value of the variable called "filename" in the file_log function to "Charge_T5_C2.csv". Then, type the command "gcc I2C.c -o I2C" to compile the script.** | **C script compiled successfully** | | **There is no error message when typing the compiling command** | | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **ETC-EPS-002-120** | **Connect the D-Sub9 Pin 1 (Vcc) with the positive cable of the Power Supply and the D-Sub9 Pin 3 (GND) with the Power Supply's ground. Every 5 min, disconnect the Power Supply wires, and then connect again the Raspberry GND and disconnect the RBF from the GND to switch ON the EPS. Execute the script (./I2C) and type "get" and then "log". and then disconnect again the Raspberry and EPS and connect the Power Supply wires.Repeat the process every 5 min during 45 min. Once done that, the test is finished.** | **Extraction of the EPS housekeeping information in charge mode and visualization of it by the command prompt during 45 min.** | | **All the data (PoL, vbatt, Temperatures,etc. ) is correctly seen in the command prompt** | | | |
| **ETC-EPS-002-130** | **Switch off the EPS by connecting the D-Sub9 Pin2 (RBF) to the DSub-9 Pin3 (GND) and disconnect the GND from the Raspberry.** | **There is no communication between the EPS and the OBC.** | | **DSub-9 Pin2 (RBF) and DSub-9 Pin3 (GND) are connected. If the get command inside the I2C script is executed, there is no information extracted.** | | | |
| **ETC-EPS-002-140** | **Heat TVAC environment and SS to 35ºC.** | **Heating of the SS.** | | **35ºC reached by the SS.** | | | |
| **ETC-EPS-002-150** | **Wait 15 min at 35ºC (dwell).** | **No temperature fluctuations during 15 min.** | | **No temperature fluctuations during 15 min.** | | | |
| **ETC-EPS-002-160** | **Start of the Functional Test in charge mode. Go to the C script (nano I2C.c) and change the value of the variable called "filename" in the file_log function to "Charge_T35_C2.csv". Then, type the command "gcc I2C.c -o I2C" to compile the script.** | **C script compiled successfully** | | **There is no error message when typing the compiling command** | | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **ETC-EPS-002-170** | **Connect the D-Sub9 Pin 1 (Vcc) with the positive cable of the Power Supply and the D-Sub9 Pin 3 (GND) with the Power Supply's ground. Every 5 min, disconnect the Power Supply wires, and then connect again the Raspberry GND and disconnect the RBF from the GND to switch ON the EPS. Execute the script (./I2C) and type "get" and then "log". and then disconnect again the Raspberry and EPS and connect the Power Supply wires.Repeat the process every 5 min during 45 min. Once done that, the test is finished.** | **Extraction of the EPS housekeeping information in charge mode and visualization of it by the command prompt during 45 min.** | | **All the data (PoL, vbatt, Temperatures,etc. ) is correctly seen in the command prompt** | | | |
| **ETC-EPS-002-180** | **Switch off the EPS by connecting the DSub-9 Pin2 (RBF) to the DSub-9 Pin3 (GND) and disconnect the GND from the Raspberry.** | **There is no communication between the EPS and the OBC.** | | **DSub-9 Pin2 (RBF) and DSub-9 Pin3 (GND) are connected. If the get command inside the I2C script is executed, there is no information extracted.** | | | |
| **ETC-EPS-002-190** | **Heat TVAC environment and SS to 50ºC.** | **Heating of the SS.** | | **50ºC reached by the SS.** | | | |
| **ETC-EPS-002-200** | **Wait 15 min at 50ºC (dwell).** | **No temperature fluctuations during 15 min.** | | **No temperature fluctuations during 15 min.** | | | |
| **ETC-EPS-002-210** | **Start of the Functional Test in the EPS. Disconnect the DSub-9 Pin 2 (RBF) from the DSub-9 Pin3 (GND) in order to switch on the EPS in discharge mode and connect the GND to the Raspberry.Connect the computer with the OBC Simulator through SSH protocol.** | **Connection between the computer and the OBC and EPS switched on** | | **We are able to send a ping from the computer to the assigned IP direction of the Raspberry** | | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **ETC-EPS-002-220** | **Execution of the C script (./I2C). Once done, type "get" in the command prompt to extract the EPS housekeeping** | **Extraction of the EPS housekeeping information and visualization of it by the command prompt** | | **All the data (PoL, vbatt, Temperatures,etc. ) is correctly seen in the command prompt.Current values match with values in Table 2 and temperatures match with the temperatures shown by the TVAC** | | | |
| **ETC-EPS-002-230** | **Type the command "register NoPoL_T50_C2.csv NoPoL" and wait 15 min.** | **Extraction of the EPS housekeeping information and visualization of it by the command prompt during 15 min without any load connected** | | **All the data (PoL, vbatt, Temperatures,etc. ) is correctly seen in the command prompt.Current values match with values in Table 2 and temperatures match with the temperatures shown by the TVAC** | | | |
| **ETC-EPS-002-240** | **Type the command "register Survival_T50_C2.csv Survival" and wait 45 min.** | **Extraction of the EPS housekeeping information and visualization of it by the command prompt during 45 min in Survival Mode (PoL 1,4,6 active and also 2 the first 9 min)** | | **All the data (PoL, vbatt, Temperatures,etc. ) is correctly seen in the command prompt.Current values match with values in Table 2 and temperatures match with the temperatures shown by the TVAC** | | | |
| **ETC-EPS-002-250** | **Type the command "register Nominal_T50_C2.csv Nominal" and wait 45 min.** | **Extraction of the EPS housekeeping information and visualization of it by the command prompt during 45 min in Nominal Mode (each PoL connected except PoL 2 that is only the first 9 min)** | | **All the data (PoL, vbatt, Temperatures,etc. ) is correctly seen in the command prompt.Current values match with values in Table 2 and temperatures match with the temperatures shown by the TVAC** | | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **ETC-EPS-002-260** | **End of the Functional Test. Switch off the EPS by connecting the DSub-9 Pin2 (RBF) to the DSub-9 Pin3 (GND) and disconnect the DSub-9 Pin3 (GND) from the Raspberry.** | **There is no communication between the EPS and the OBC.** | | **DSub-9 Pin2 (RBF) and DSub-9 Pin3 (GND) are connected. If the get command inside the I2C script is executed, there is no information extracted.** | | | |
| **ETC-EPS-002-270** | **Return SS and TVAC to ambient temperature** | **Tambient is reached** | | **Temperature rises in the SS** | | | |
| **END OF CYCLE 2** | | | | | | | |
| | | | | | | | |
| **START OF CYCLE 3** | | | | | | | |
| **ETC-EPS-003-010** | **Cool TVAC environment and SS to 0ºC** | **Cooling of the SS.** | | **0ºC reached by the SS.** | | | |
| **ETC-EPS-003-020** | **Wait 15 min at 0ºC (dwell).** | **No temperature fluctuations during 15 min.** | | **No temperature fluctuations during 15 min.** | | | |
| **ETC-EPS-003-030** | **Start of the Functional Test in the EPS. Disconnect the DSub-9 Pin2 (RBF) from the DSub-9 Pin3 (GND) in order to switch on the EPS in discharge mode and connect the GND to the Raspberry. Connect the computer with the OBC Simulator through SSH protocol.** | **Connection between the computer and the OBC and EPS switched on** | | **We are able to send a ping from the computer to the assigned IP direction of the Raspberry.** | | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **ETC-EPS-003-040** | **Execution of the C script (./I2C). Once done, type "get" in the command prompt to extract the EPS housekeeping** | **Extraction of the EPS housekeeping information and visualization of it by the command prompt** | | **All the data (PoL, vbatt, Temperatures,etc. ) is correctly seen in the command prompt.Current values match with values in Table 2 and temperatures match with the temperatures shown by the TVAC** | | | |
| **ETC-EPS-003-050** | **Type the command "register NoPoL_T0_C3.csv NoPoL" and wait 15 min.** | **Extraction of the EPS housekeeping information and visualization of it by the command prompt during 15 min without any load connected** | | **All the data (PoL, vbatt, Temperatures,etc. ) is correctly seen in the command prompt.Current values match with values in Table 2 and temperatures match with the temperatures shown by the TVAC** | | | |
| **ETC-EPS-003-060** | **Type the command "register Survival_T0_C3.csv Survival" and wait 45 min.** | **Extraction of the EPS housekeeping information and visualization of it by the command prompt during 45 min in Survival Mode (PoL 1,4,6 active and also 2 the first 9 min)** | | **All the data (PoL, vbatt, Temperatures,etc. ) is correctly seen in the command prompt.Current values match with values in Table 2 and temperatures match with the temperatures shown by the TVAC** | | | |
| **ETC-EPS-003-070** | **Type the command "register Nominal_T0_C3.csv Nominal" and wait 45 min.** | **Extraction of the EPS housekeeping information and visualization of it by the command prompt during 45 min in Nominal Mode (each PoL connected except PoL 2 that is only the first 9 min)** | | **All the data (PoL, vbatt, Temperatures,etc. ) is correctly seen in the command prompt.Current values match with values in Table 2 and temperatures match with the temperatures shown by the TVAC** | | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **ETC-EPS-003-080** | **End of the Functional Test. Switch off the EPS by connecting the DSub-9 Pin2 (RBF) to the DSub-9 Pin3 (GND) and disconnect the DSub-9 Pin3 (GND) from the Raspberry.** | **There is no communication between the EPS and the OBC.** | | **DSub-9 Pin2 (RBF) and DSub-9 Pin3 (GND) are connected. If the get command inside the I2C script is executed, there is no information extracted.** | | | |
| **ETC-EPS-003-090** | **Heat TVAC environment and SS to 5ºC.** | **Heating of the SS.** | | **5ºC reached by the SS.** | | | |
| **ETC-EPS-003-100** | **Wait 15 min at 5ºC (dwell).** | **No temperature fluctuations during 15 min.** | | **No temperature fluctuations during 15 min.** | | | |
| **ETC-EPS-003-110** | **Start of the Functional Test in charge mode. Go to the C script (nano I2C.c) and change the value of the variable called "filename" in the file_log function to "Charge_T5_C3.csv". Then, type the command "gcc I2C.c -o I2C" to compile the script.** | **C script compiled successfully** | | **There is no error message when typing the compiling command** | | | |
| **ETC-EPS-003-120** | **Connect the D-Sub9 Pin 1 (Vcc) with the positive cable of the Power Supply and the D-Sub9 Pin 3 (GND) with the Power Supply's ground. Every 5 min, disconnect the Power Supply wires, and then connect again the Raspberry GND and disconnect the RBF from the GND to switch ON the EPS. Execute the script (./I2C) and type "get" and then "log". and then disconnect again the Raspberry and EPS and connect the Power Supply wires. Repeat the process every 5 min during 45 min. Once done that, the test is finished.** | **Extraction of the EPS housekeeping information in charge mode and visualization of it by the command prompt during 45 min.** | | **All the data (PoL, vbatt, Temperatures,etc. ) is correctly seen in the command prompt** | | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **ETC-EPS-003-130** | **Switch off the EPS by connecting the D-Sub9 Pin2 (RBF) to the DSub-9 Pin3 (GND) and disconnect the GND from the Raspberry.** | **There is no communication between the EPS and the OBC.** | | **DSub-9 Pin2 (RBF) and DSub-9 Pin3 (GND) are connected. If the get command inside the I2C script is executed, there is no information extracted.** | | | |
| **ETC-EPS-003-140** | **Heat TVAC environment and SS to 35ºC.** | **Heating of the SS.** | | **35ºC reached by the SS.** | | | |
| **ETC-EPS-003-150** | **Wait 15 min at 35ºC (dwell).** | **No temperature fluctuations during 15 min.** | | **No temperature fluctuations during 15 min.** | | | |
| **ETC-EPS-003-160** | **Start of the Functional Test in charge mode. Go to the C script (nano I2C.c) and change the value of the variable called "filename" in the file_log function to "Charge_T35_C3.csv". Then, type the command "gcc I2C.c -o I2C" to compile the script.** | **C script compiled successfully** | | **There is no error message when typing the compiling command** | | | |
| **ETC-EPS-003-170** | **Connect the D-Sub9 Pin 1 (Vcc) with the positive cable of the Power Supply and the D-Sub9 Pin 3 (GND) with the Power Supply's ground. Every 5 min, disconnect the Power Supply wires, and then connect again the Raspberry GND and disconnect the RBF from the GND to switch ON the EPS. Execute the script (./I2C) and type "get" and then "log". and then disconnect again the Raspberry and EPS and connect the Power Supply wires.Repeat the process every 5 min during 45 min. Once done that, the test is finished.** | **Extraction of the EPS housekeeping information in charge mode and visualization of it by the command prompt during 45 min.** | | **All the data (PoL, vbatt, Temperatures,etc. ) is correctly seen in the command prompt** | | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **ETC-EPS-003-180** | **Switch off the EPS by connecting the DSub-9 Pin2 (RBF) to the DSub-9 Pin3 (GND) and disconnect the GND from the Raspberry.** | **There is no communication between the EPS and the OBC.** | | **DSub-9 Pin2 (RBF) and DSub-9 Pin3 (GND) are connected. If the get command inside the I2C script is executed, there is no information extracted.** | | | |
| **ETC-EPS-003-190** | **Heat TVAC environment and SS to 45ºC.** | **Heating of the SS.** | | **45ºC reached by the SS.** | | | |
| **ETC-EPS-003-200** | **Wait 15 min at 45ºC (dwell).** | **No temperature fluctuations during 15 min.** | | **No temperature fluctuations during 15 min.** | | | |
| **ETC-EPS-003-210** | **Start of the Functional Test in the EPS. Disconnect the DSub-9 Pin 2 (RBF) from the DSub-9 Pin3 (GND) in order to switch on the EPS in discharge mode and connect the GND to the Raspberry.Connect the computer with the OBC Simulator through SSH protocol.** | **Connection between the computer and the OBC and EPS switched on** | | **We are able to send a ping from the computer to the assigned IP direction of the Raspberry** | | | |
| **ETC-EPS-003-220** | **Execution of the C script (./I2C). Once done, type "get" in the command prompt to extract the EPS housekeeping** | **Extraction of the EPS housekeeping information and visualization of it by the command prompt** | | **All the data (PoL, vbatt, Temperatures,etc. ) is correctly seen in the command prompt.Current values match with values in Table 2 and temperatures match with the temperatures shown by the TVAC** | | | |
| **ETC-EPS-003-230** | **Type the command "register NoPoL_T45_C3.csv NoPoL" and wait 15 min.** | **Extraction of the EPS housekeeping information and visualization of it by the command prompt during 15 min without any load connected** | | **All the data (PoL, vbatt, Temperatures,etc. ) is correctly seen in the command prompt.Current values match with values in Table 2 and temperatures match with the temperatures shown by the TVAC** | | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **ETC-EPS-003-240** | **Type the command "register Survival_T45_C3.csv Survival" and wait 45 min.** | **Extraction of the EPS housekeeping information and visualization of it by the command prompt during 45 min in Survival Mode (PoL 1,4,6 active and also 2 the first 9 min)** | | **All the data (PoL, vbatt, Temperatures,etc. ) is correctly seen in the command prompt.Current values match with values in Table 2 and temperatures match with the temperatures shown by the TVAC** | | | |
| **ETC-EPS-003-250** | **Type the command "register Nominal_T45_C3.csv Nominal" and wait 45 min.** | **Extraction of the EPS housekeeping information and visualization of it by the command prompt during 45 min in Nominal Mode (each PoL connected except PoL 2 that is only the first 9 min)** | | **All the data (PoL, vbatt, Temperatures,etc. ) is correctly seen in the command prompt.Current values match with values in Table 2 and temperatures match with the temperatures shown by the TVAC** | | | |
| **ETC-EPS-003-260** | **End of the Functional Test. Switch off the EPS by connecting the DSub-9 Pin2 (RBF) to the DSub-9 Pin3 (GND) and disconnect the DSub-9 Pin3 (GND) from the Raspberry.** | **There is no communication between the EPS and the OBC.** | | **DSub-9 Pin2 (RBF) and DSub-9 Pin3 (GND) are connected. If the get command inside the I2C script is executed, there is no information extracted.** | | | |
| **ETC-EPS-003-270** | **Return SS and TVAC to ambient temperature** | **Tambient is reached** | | **Temperature rises in the SS** | | | |
| **END OF CYCLE 3** | | | | | | | |
| | | | | | | | |
| **START OF CYCLE 4** | | | | | | | |
| **ETC-EPS-004-010** | **Cool TVAC environment and SS to 0ºC** | **Cooling of the SS.** | | **0ºC reached by the SS.** | | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **ETC-EPS-004-020** | **Wait 15 min at 0ºC (dwell).** | **No temperature fluctuations during 15 min.** | | **No temperature fluctuations during 15 min.** | | | |
| **ETC-EPS-004-030** | **Start of the Functional Test in the EPS. Disconnect the DSub-9 Pin2 (RBF) from the DSub-9 Pin3 (GND) in order to switch on the EPS in discharge mode and connect the GND to the Raspberry.Connect the computer with the OBC Simulator through SSH protocol.** | **Connection between the computer and the OBC and EPS switched on** | | **We are able to send a ping from the computer to the assigned IP direction of the Raspberry.** | | | |
| **ETC-EPS-004-040** | **Execution of the C script (./I2C). Once done, type "get" in the command prompt to extract the EPS housekeeping** | **Extraction of the EPS housekeeping information and visualization of it by the command prompt** | | **All the data (PoL, vbatt, Temperatures,etc. ) is correctly seen in the command prompt.Current values match with values in Table 2 and temperatures match with the temperatures shown by the TVAC** | | | |
| **ETC-EPS-004-050** | **Type the command "register NoPoL_T0_C4.csv NoPoL" and wait 15 min.** | **Extraction of the EPS housekeeping information and visualization of it by the command prompt during 15 min without any load connected** | | **All the data (PoL, vbatt, Temperatures,etc. ) is correctly seen in the command prompt.Current values match with values in Table 2 and temperatures match with the temperatures shown by the TVAC** | | | |
| **ETC-EPS-004-060** | **Type the command "register Survival_T0_C4.csv Survival" and wait 45 min.** | **Extraction of the EPS housekeeping information and visualization of it by the command prompt during 45 min in Survival Mode (PoL 1,4,6 active and also 2 the first 9 min)** | | **All the data (PoL, vbatt, Temperatures,etc. ) is correctly seen in the command prompt.Current values match with values in Table 2 and temperatures match with the temperatures shown by the TVAC** | | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **ETC-EPS-004-070** | **Type the command "register Nominal_T0_C4.csv Nominal" and wait 45 min.** | **Extraction of the EPS housekeeping information and visualization of it by the command prompt during 45 min in Nominal Mode (each PoL connected except PoL 2 that is only the first 9 min)** | | **All the data (PoL, vbatt, Temperatures,etc. ) is correctly seen in the command prompt.Current values match with values in Table 2 and temperatures match with the temperatures shown by the TVAC** | | | |
| **ETC-EPS-004-080** | **End of the Functional Test. Switch off the EPS by connecting the DSub-9 Pin2 (RBF) to the DSub-9 Pin3 (GND) and disconnect the DSub-9 Pin3 (GND) from the Raspberry.** | **There is no communication between the EPS and the OBC.** | | **DSub-9 Pin2 (RBF) and DSub-9 Pin3 (GND) are connected. If the get command inside the I2C script is executed, there is no information extracted.** | | | |
| **ETC-EPS-004-090** | **Heat TVAC environment and SS to 5ºC.** | **Heating of the SS.** | | **5ºC reached by the SS.** | | | |
| **ETC-EPS-004-100** | **Wait 15 min at 5ºC (dwell).** | **No temperature fluctuations during 15 min.** | | **No temperature fluctuations during 15 min.** | | | |
| **ETC-EPS-004-110** | **Start of the Functional Test in charge mode. Go to the C script (nano I2C.c) and change the value of the variable called "filename" in the file_log function to "Charge_T5_C4.csv". Then, type the command "gcc I2C.c -o I2C" to compile the script.** | **C script compiled successfully** | | **There is no error message when typing the compiling command** | | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **ETC-EPS-004-120** | **Connect the D-Sub9 Pin 1 (Vcc) with the positive cable of the Power Supply and the D-Sub9 Pin 3 (GND) with the Power Supply's ground. Every 5 min, disconnect the Power Supply wires, and then connect again the Raspberry GND and disconnect the RBF from the GND to switch ON the EPS. Execute the script (./I2C) and type "get" and then "log". and then disconnect again the Raspberry and EPS and connect the Power Supply wires. Repeat the process every 5 min during 45 min. Once done that, the test is finished.** | **Extraction of the EPS housekeeping information in charge mode and visualization of it by the command prompt during 45 min.** | | **All the data (PoL, vbatt, Temperatures,etc. ) is correctly seen in the command prompt** | | | |
| **ETC-EPS-004-130** | **Switch off the EPS by connecting the D-Sub9 Pin2 (RBF) to the DSub-9 Pin3 (GND) and disconnect the GND from the Raspberry.** | **There is no communication between the EPS and the OBC.** | | **DSub-9 Pin2 (RBF) and DSub-9 Pin3 (GND) are connected. If the get command inside the I2C script is executed, there is no information extracted.** | | | |
| **ETC-EPS-004-140** | **Heat TVAC environment and SS to 35ºC.** | **Heating of the SS.** | | **35ºC reached by the SS.** | | | |
| **ETC-EPS-004-150** | **Wait 15 min at 35ºC (dwell).** | **No temperature fluctuations during 15 min.** | | **No temperature fluctuations during 15 min.** | | | |
| **ETC-EPS-004-160** | **Start of the Functional Test in charge mode. Go to the C script (nano I2C.c) and change the value of the variable called "filename" in the file_log function to "Charge_T35_C4.csv". Then, type the command "gcc I2C.c -o I2C" to compile the script.** | **C script compiled successfully** | | **There is no error message when typing the compiling command** | | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **ETC-EPS-004-170** | **Connect the D-Sub9 Pin 1 (Vcc) with the positive cable of the Power Supply and the D-Sub9 Pin 3 (GND) with the Power Supply's ground. Every 5 min, disconnect the Power Supply wires, and then connect again the Raspberry GND and disconnect the RBF from the GND to switch ON the EPS. Execute the script (./I2C) and type "get" and then "log". and then disconnect again the Raspberry and EPS and connect the Power Supply wires. Repeat the process every 5 min during 45 min. Once done that, the test is finished.** | **Extraction of the EPS housekeeping information in charge mode and visualization of it by the command prompt during 45 min.** | | **All the data (PoL, vbatt, Temperatures,etc. ) is correctly seen in the command prompt** | | | |
| **ETC-EPS-004-180** | **Switch off the EPS by connecting the DSub-9 Pin2 (RBF) to the DSub-9 Pin3 (GND) and disconnect the GND from the Raspberry.** | **There is no communication between the EPS and the OBC.** | | **DSub-9 Pin2 (RBF) and DSub-9 Pin3 (GND) are connected. If the get command inside the I2C script is executed, there is no information extracted.** | | | |
| **ETC-EPS-004-190** | **Heat TVAC environment and SS to 45ºC.** | **Heating of the SS.** | | **45ºC reached by the SS.** | | | |
| **ETC-EPS-004-200** | **Wait 15 min at 45ºC (dwell).** | **No temperature fluctuations during 15 min.** | | **No temperature fluctuations during 15 min.** | | | |
| **ETC-EPS-004-210** | **Start of the Functional Test in the EPS. Disconnect the DSub-9 Pin 2 (RBF) from the DSub-9 Pin3 (GND) in order to switch on the EPS in discharge mode and connect the GND to the Raspberry. Connect the computer with the OBC Simulator through SSH protocol.** | **Connection between the computer and the OBC and EPS switched on** | | **We are able to send a ping from the computer to the assigned IP direction of the Raspberry** | | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **ETC-EPS-004-220** | **Execution of the C script (./I2C). Once done, type "get" in the command prompt to extract the EPS housekeeping** | **Extraction of the EPS housekeeping information and visualization of it by the command prompt** | | **All the data (PoL, vbatt, Temperatures,etc. ) is correctly seen in the command prompt.Current values match with values in Table 2 and temperatures match with the temperatures shown by the TVAC** | | | |
| **ETC-EPS-004-230** | **Type the command "register NoPoL_T45_C4.csv NoPoL" and wait 15 min.** | **Extraction of the EPS housekeeping information and visualization of it by the command prompt during 15 min without any load connected** | | **All the data (PoL, vbatt, Temperatures,etc. ) is correctly seen in the command prompt.Current values match with values in Table 2 and temperatures match with the temperatures shown by the TVAC** | | | |
| **ETC-EPS-004-240** | **Type the command "register Survival_T45_C4.csv Survival" and wait 45 min.** | **Extraction of the EPS housekeeping information and visualization of it by the command prompt during 45 min in Survival Mode (PoL 1,4,6 active and also 2 the first 9 min)** | | **All the data (PoL, vbatt, Temperatures,etc. ) is correctly seen in the command prompt.Current values match with values in Table 2 and temperatures match with the temperatures shown by the TVAC** | | | |
| **ETC-EPS-004-250** | **Type the command "register Nominal_T45_C4.csv Nominal" and wait 45 min.** | **Extraction of the EPS housekeeping information and visualization of it by the command prompt during 45 min in Nominal Mode (each PoL connected except PoL 2 that is only the first 9 min)** | | **All the data (PoL, vbatt, Temperatures,etc. ) is correctly seen in the command prompt.Current values match with values in Table 2 and temperatures match with the temperatures shown by the TVAC** | | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **ETC-EPS-004-260** | **End of the Functional Test. Switch off the EPS by connecting the DSub-9 Pin2 (RBF) to the DSub-9 Pin3 (GND) and disconnect the DSub-9 Pin3 (GND) from the Raspberry.** | **There is no communication between the EPS and the OBC.** | | **DSub-9 Pin2 (RBF) and DSub-9 Pin3 (GND) are connected. If the get command inside the I2C script is executed, there is no information extracted.** | | | |
| **ETC-EPS-004-270** | **Return SS and TVAC to ambient temperature** | **Tambient is reached** | | **Temperature rises in the SS** | | | |
| **ETC-EPS-004-280** | **Pressurize TVAC to ambient pressure.** | **Pressure rises** | | **Ambient pressure is reached inside vacuum** | | | |
| **ETC-EPS-004-290** | **Open TVAC** | **TVAC opens** | | **TVAC opens** | | | |
| **ETC-EPS-004-300** | **Disconnect thermocouples and internal connections from the DSub9.** | **Nothing connects the TVAC to the SS** | | **All disconnection doesn't break any connector** | | | |
| **ETC-EPS-004-310** | **Detach EPS from MGSE.** | **Detachment of the EPS from the MGSE** | | **EPS independent from the MGSE** | | | |
| **END OF TEST** | | | | | | | |

**Table 7 - Step-by-Step procedure for the test**

## 8.6. EPS TRPT

NANOSAT LAB

# Test Report

Electrical Power Subsystem Environmental Test Campaign

*University Name:* Universitat Politècnica de Catalunya (UPC)
Campus Nord, building D3
08034 Barcelona, SPAIN
Date: 02/08/2020
*Reference:* 3C4_TRPT_20190125_v1.1
*Distribution List:*

| Destination | Contact |
|---|---|
| [3]Cat-4 Team | 3cat4@tsc.upc.edu |

# Authors


| First Name | Last Name | Contact | Role |
|---|---|---|---|
| Juanjo | Medina Musellas | juanjomedinamusellas@gmail.com | -- |
| Albert | Rodríguez Casellas | albert.rodriguez.casellas@gmail.com | -- |


# Revised by

| First Name | Last Name | Contact | Role |
|---|---|---|---|
| Lara | Fernandez Capon | lara-pilar.fernandez@tsc.upc.edu | System Engineer |

# Change Log

| Revision | Release date | Section | Description |
|---|---|---|---|
| V1.0 | 28/02/2019 | Whole document | First release |
| | | | |

# Applicable Documents

| AD# | Reference | Revision |
|---|---|---|
| [AD1] | TSTP_EPS_20200202 | v1.2 |

# List of Abbreviations

| | |
|---|---|
| EPS | Electrical Power Subsystem |
| MGSE | Mechanical Ground Support Equipment |
| PoL | Point of Load |
| SS | Standalone Subsystem |
| TVAC | Thermal Vacuum Chamber |

# Table of contents

# List of Figures

# List of Tables

# Introduction

This document aims to contain the results and the consequent conclusions of the test performed following the TSTP that is included in Applicable documents [AD1]. The test report includes the anomalies that may have happened during the test performance. The conclusions extracted from the plots obtained performing the TVAC test are shown in the synthesis section. In order to understand the hypothesis made, the reader should have some basic knowledge about thermodynamics, in concrete, heat transfer and some bases about electricity and electronics.

# Test Report

## EPS Acceptance Test

The Electrical Power Subsystem (EPS) Acceptance Test consists in qualifying the subsystem performance under some temperature conditions.
Concretely, the test performs 4 cycles. The temperature range of each cycle, specified within the TRTP_EPS_20200202_v1.2, is defined in Figure 1.
Table 1 reflects the different temperatures accounted for the definition of the temperature range of the test.

| EPS Subsystem Mode | Batteries Operational Temperature Range (ºC) | | Design Temperature Range (ºC) | | Mission Temperature Range (ºC) | | Mission Acceptance Temperature Range (ºC) | | Subsystem Acceptance Temperature Range (ºC) | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max |
| Discharge Mode | -10 | 60 | 0 | 50 | 5 | 40 | 0 | 45 | 0 | 50 |
| Charge Mode | -5 | 45 | 5 | 35 | 10 | 30 | 5 | 35 | 5 | 35 |

**Table 1.- Acceptance and design temperature range**

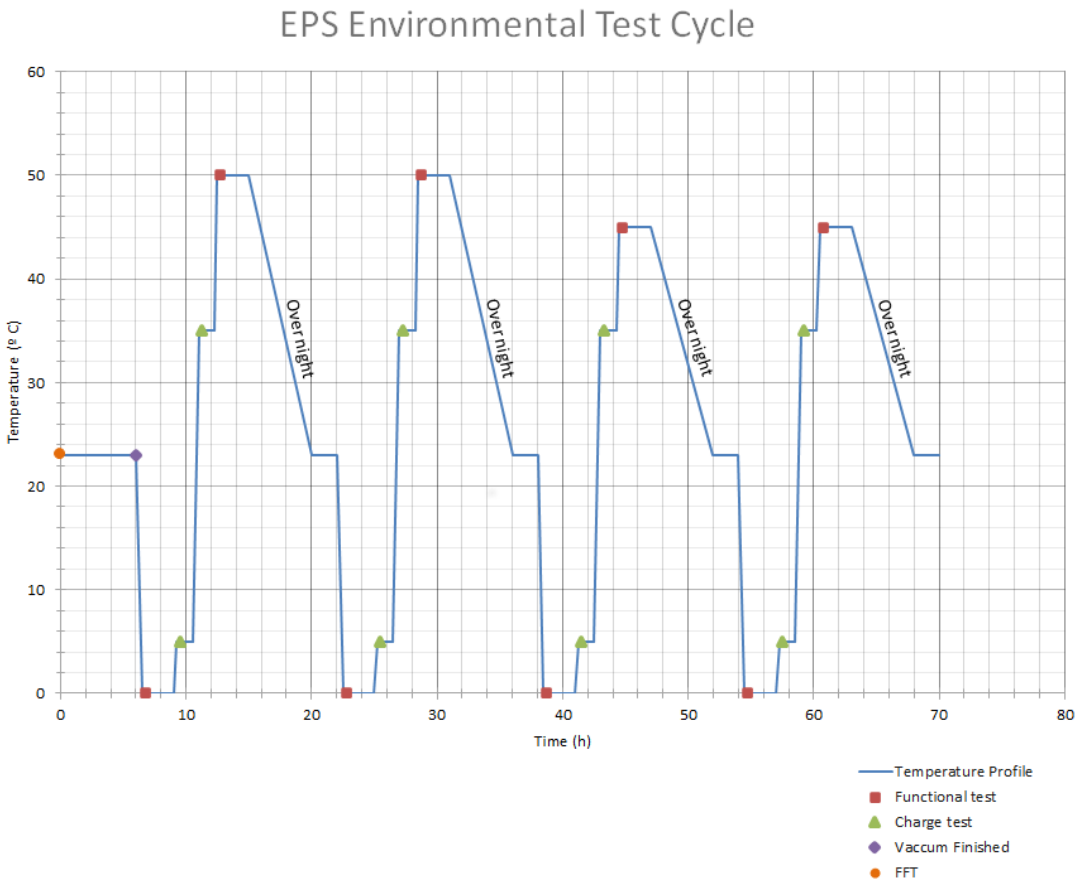



**Figure 1 - Thermal profile of EPS Subsystem Acceptance**

| Execution Date | Facility used | Operators |
|---|---|---|
| 15/07/2020<br>16/07/2020<br>20/07/2020<br>23/07/2020<br>24/07/2020<br>27/07/2020<br>28/07/2020<br>30/07/2020<br>31/07/2020<br>03/08/2020 | UPC NanoSat Lab Cleanroom | Juanjo Medina Musellas<br>Albert Rodríguez Casellas |

**Table 2 - Subsystem Acceptance Environment**

## Test Logbooks

This section contains every step carried out during the test procedure. Every step is measured in time and specifically explained in order to fully understand the method followed.

15/07 07:05 - EPS assembled into the CubeSat structure.

15/07 07:40 - Resistors perfboard connected to the EPS PoLs. We have coated the perfboard part which is to be in contact with the Aluminium Box in order not to produce electrical conduction.

15/07 07:42 - EPS encapsulated into the Aluminium Box.

15/07 07:43 - Internal DSub-9 pinout connected to the EPS.

15/07 07:45 - Thermocouples installed.

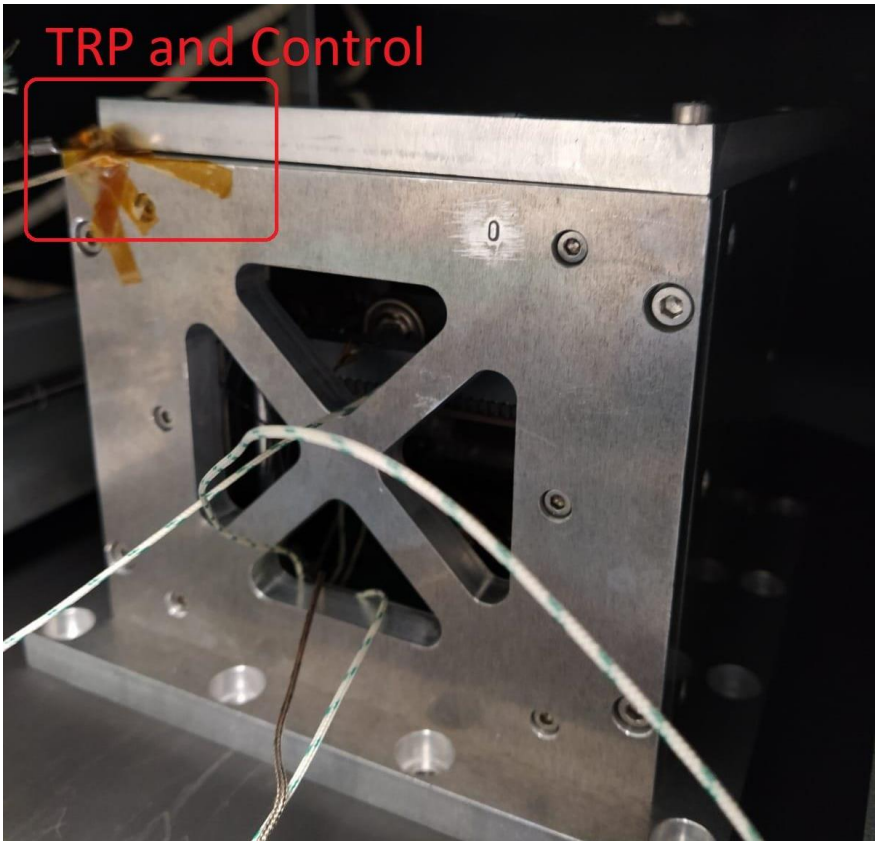



**Figure 2 - TRP and Control locations**

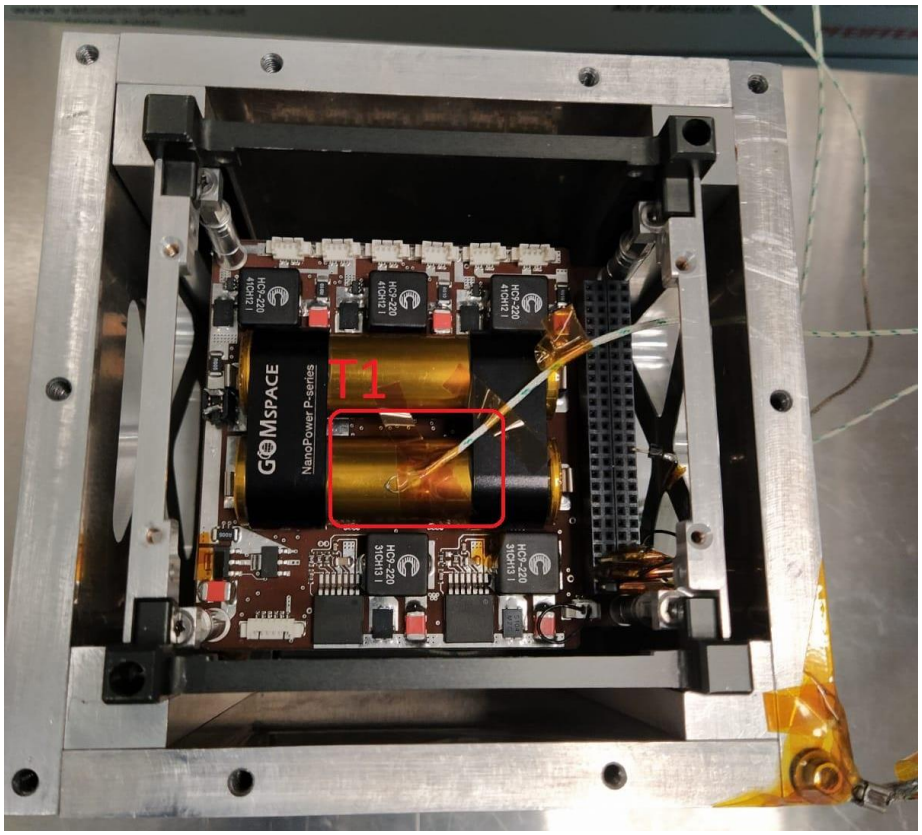



**Figure 3 - T1 location**

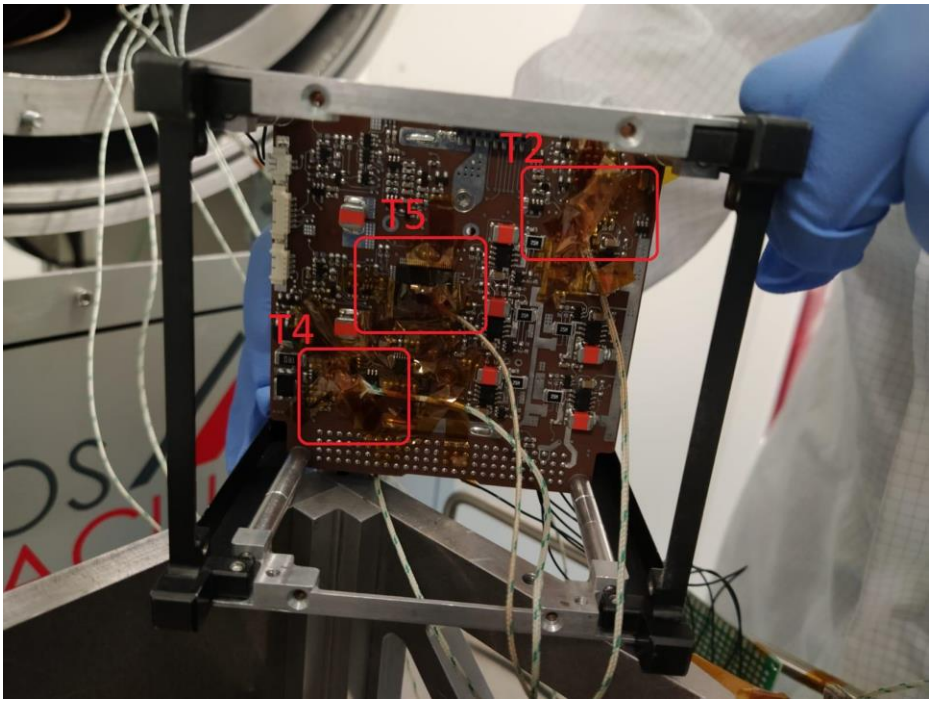



**Figure 4 - T2, T4 and T5 locations**

15/07 08:00 - EPS firmly placed into the TVAC

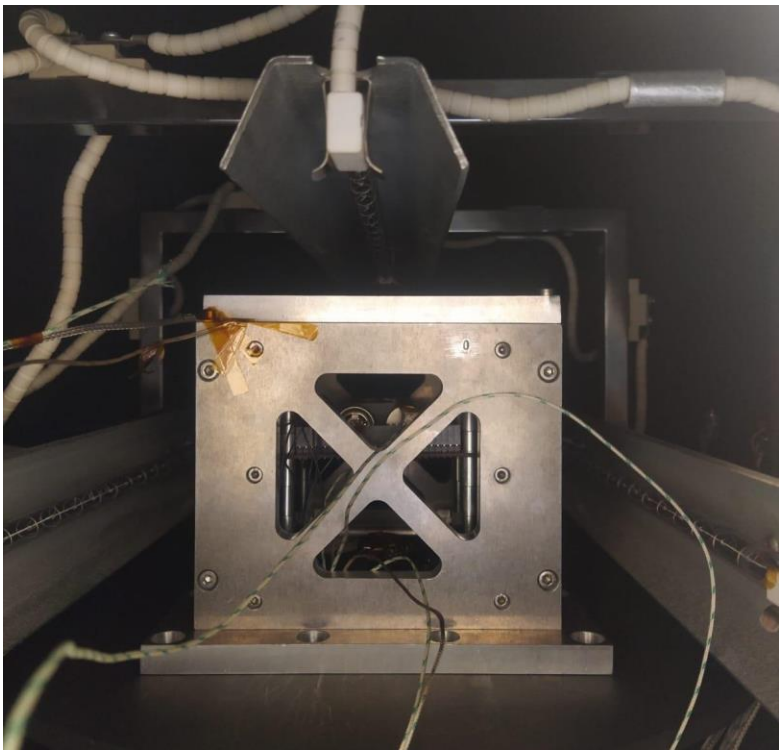


**Figure 5 - Subsystem located inside the TVAC**

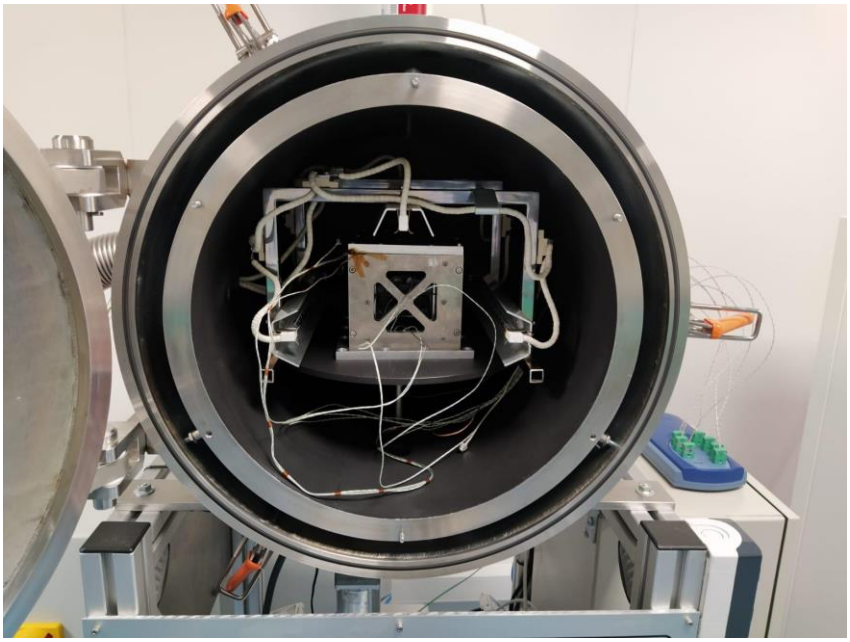


**Figure 6 - Subsystem located inside the TVAC seen from outside**

15/07 8:01 - All the necessary connections to proceed with the test are plugged.

15/07 8:03 - Start of the FFT. In this case, the Raspberry's IP address is 10.4.33.133, so the command to connect the Raspberry to the laptop is ssh pi@10.4.33.133.

15/07 9:12 - End of the FFT. Every test has been performed successfully.

15/07 10:01 - Start of the TVAC decompression. It will last until tomorrow, so the first cycle is going to start tomorrow.

16/07 9:40 - Start of the first cycle. We cool the SS to 0ºC.

16/07 13:04 - End of the first Discharge test at 0ºC. Every test performed successfully. We have seen that although consuming a considerable amount of power, there is not a remarkable increase in the temperature while doing the test. The dwell has lasted more than what we expected, which means that we have waited more time than expected so that every part of the EPS could get a homogeneous temperature.

16/07 14:41 - End of the first Charge test at 5ºC. As can be seen in the plots presented in the Analysis part, there has been a higher increase in temperature in the Charge test than in the Discharge.

16/07 19:45 - End of the first Charge test at 35ºC. Same behaviour as before. Despite trying to avoid the charge effects, there is a remarkable increase in temperature.

16/07 23:26 - End of the first Discharge test at 50ºC. The batteries' temperature sensor has approached dangerously to the operational temperature, which is 60ºC. We will have to be cautious in the upcoming tests at this temperature.

Owing to the amount of time that the first cycle has required, we have decided to divide the following cycles into two parts (cold and hot).

20/07 9:05 - Start of the hot part of the second cycle. Heating the SS until a 35ºC temperature is reached.

20/07 11:15 - End of the second Charge Test at 35ºC.

20/07 15:03 - End of the second Discharge Test at 50ºC.

20/07 15:05 - End of the hot part of the second cycle. We are going to cool the system until ambient temperature is reached so the following day we can start from ambient temperature.

23/07 11:05 - Start of the cold part of the second cycle. Cooling down the SS until a 0ºC temperature is reached.

23/07 17:13 - End of the second Discharge Test at 0ºC. There has been a huge dwell period in order to get every temperature sensor to 0ºC.

23/07 19:35 - End of the second Discharge Test at 5ºC and the second cycle. Heating the system to ambient temperature.

24/07 9:22 - Start of the hot part of the third cycle. Heating the system to 35ªC.

24/07 11:46 - End of the third Charge Test at 35ºC.

24/07 16:26 - End of the first Discharge Test at 45ºC.

24/07 16:28 - End of the hot part of the third cycle. Cooling down the SS to ambient temperature.

27/07 9:07 - Start of the cold part of the third cycle. Cooling down the SS to 0ºC.

27/07 13:13 - End of the third Discharge Test at 0ºC.

27/07 14:50 - End of the third Charge Test at 5ºC and end of the third cycle. Heating the SS to ambient temperature.

28/07 9:35 - Start of the hot part of the fourth and the last cycle. Heating the SS to 35ºC.

28/07 11:45 - End of the fourth and last Charge Test at 35ºC.

28/07 15:50 - End of the second and last Discharge Test at 45ºC.

28/07 15:52 - End of the hot part of the fourth and the last cycle. Cooling down the SS to ambient temperature.

30/07 9:31 - Start of the cold part of the fourth and last cycle. Cooling down the SS to 0ºC.

30/07 14:10 - End of the fourth and last Discharge Test at 0ºC.

30/07 15:11 - End of the fourth and last Charge Test at 5ºC and the fourth cycle. Heating the SS to ambient temperature.

31/07 9:15 - The SS is at ambient temperature, so we start the pressurisation.

03/08 9:24 - The SS is at ambient temperature and ambient pressure, so we can open the TVAC.

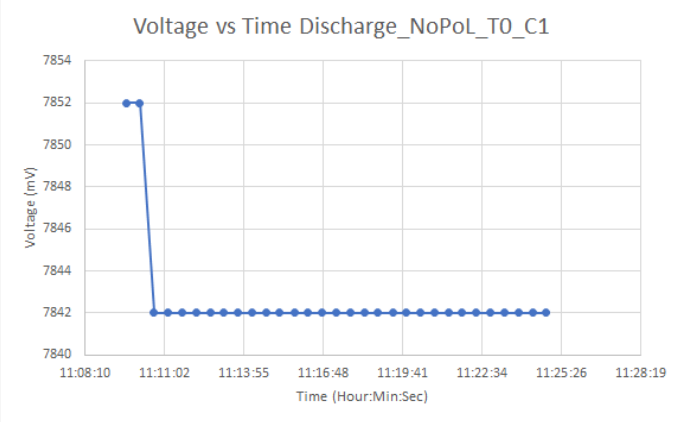
03/08 9:37 - We separate the EPS from the MGSE, which is the Aluminium Box and the CubeSat structure.

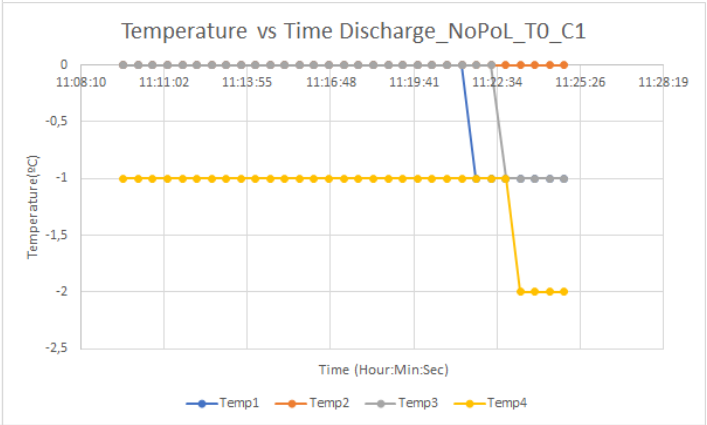
03/08 9:40 - End of the Test.

## Analysis

In this section, we will be analyzing the different results extracted from the performed test. Each cycle has four plateaus, the discharge plateau (which consists of No Points of Load, Survival and Nominal tests) and the charge plateau (which only has the Charge test). Every test gives information about voltage vs time and temperature vs time, so the overall number of plots will be 64. In each plot we will calculate the rate of discharge for the first 9 minutes (with the magnetorquers active) and for the rest of time (without them). However, the cold discharge tests will contain information about the voltage loss, not the rate of discharge, because its behavior is parabolic, not linear. The information will be presented as follows:

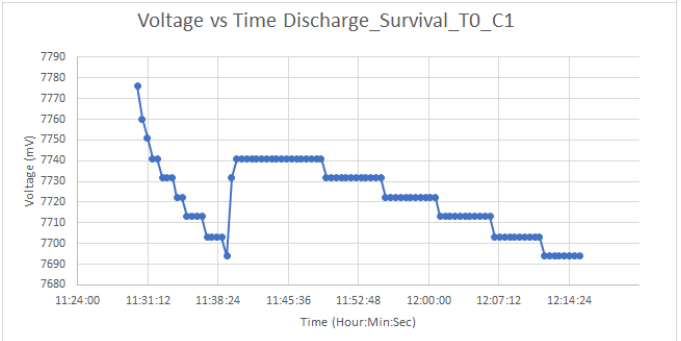
- 1st Cycle
  - 0ºC Discharge
    - NoPoL

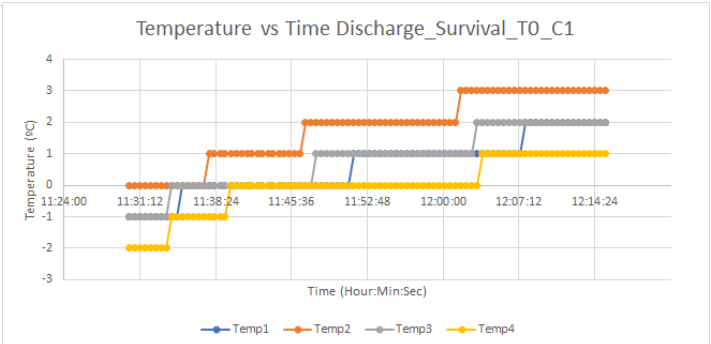



    - Survival

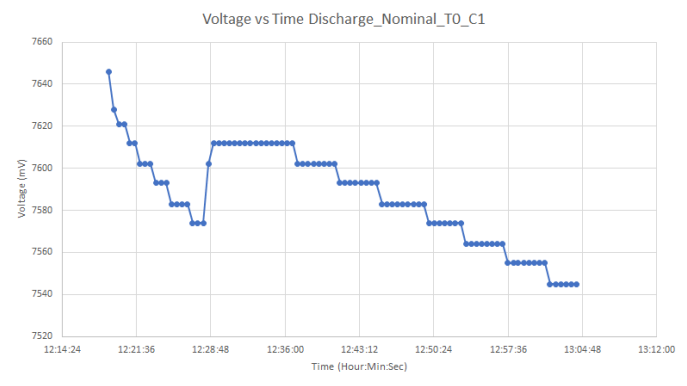
Voltage loss in the 1st section: 80 mV
Discharge rate 2nd section: 75 mV/h

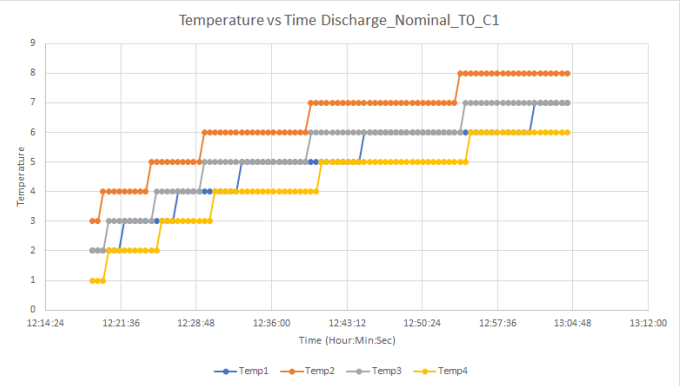


- Nominal

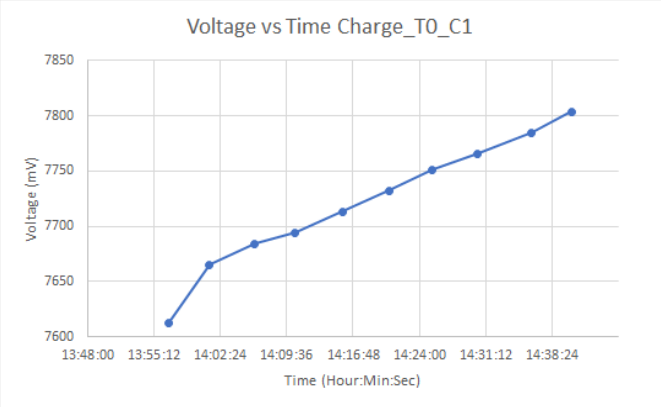
Voltage loss in the 1st section: 70 mV
Discharge rate 2nd section: 108.33 mV/h

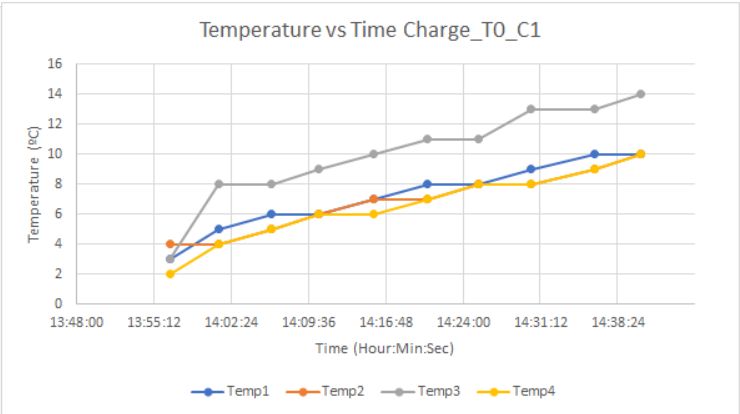



- 5ºC Charge

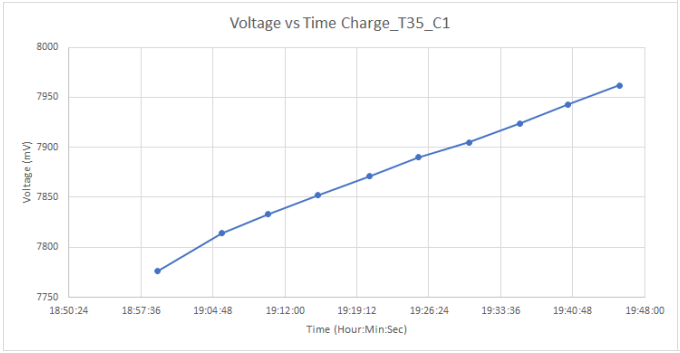
Charge rate : 253.33 mV/h

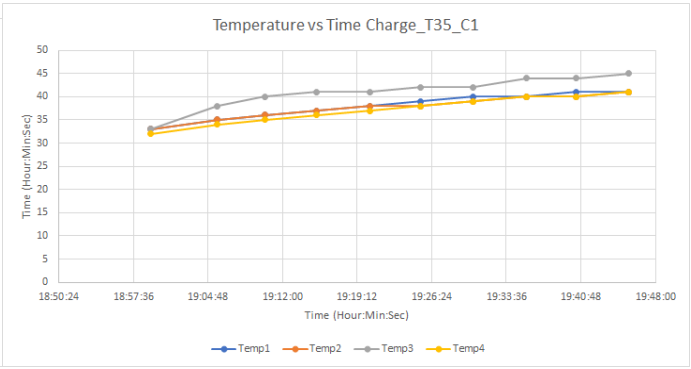


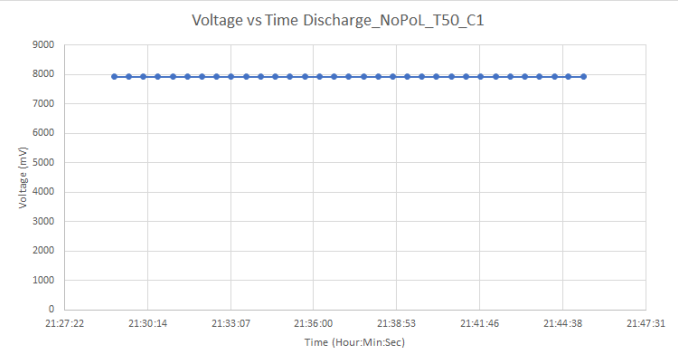

- 35ºC Charge

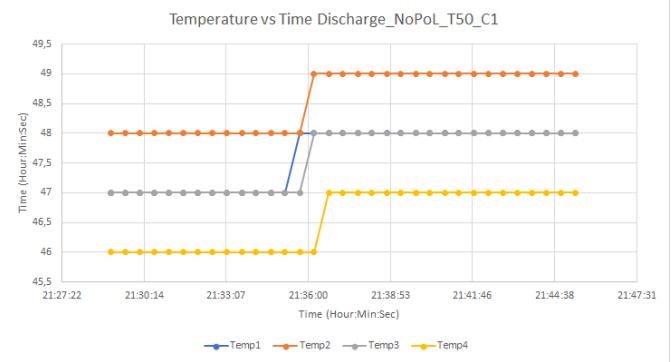
Charge rate : 246.67 mV/h



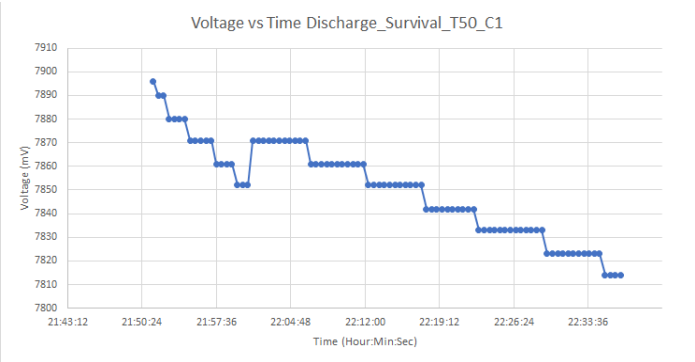
- 50ºC Discharge
  - NoPoL

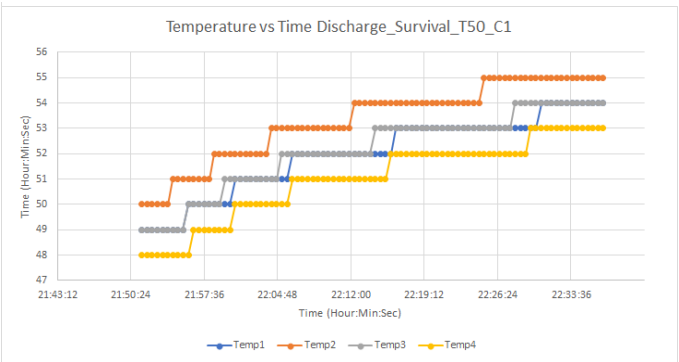



  - Survival

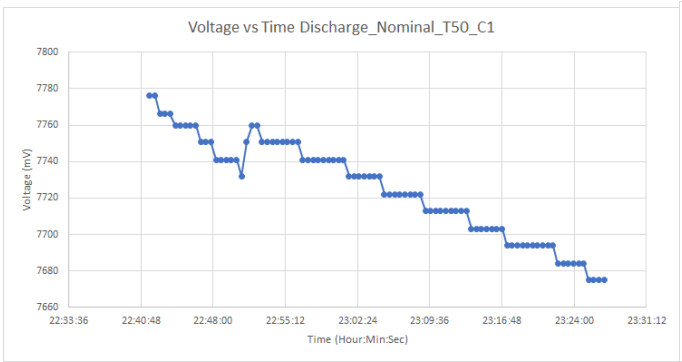
Discharge rate 1st section: 300 mV/h
Discharge rate 2nd section: 91.67 mV/h

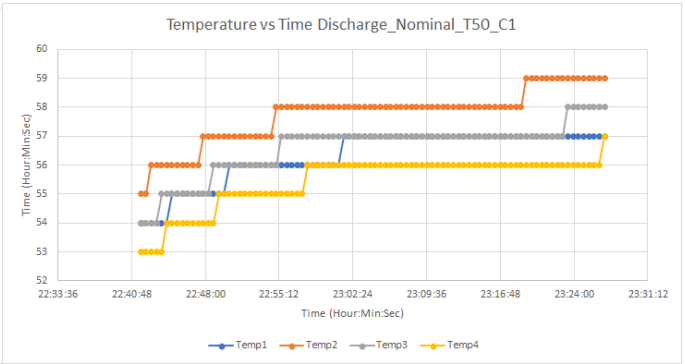


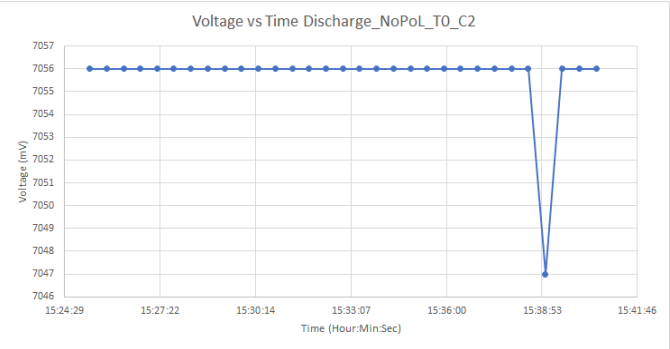

  - Nominal

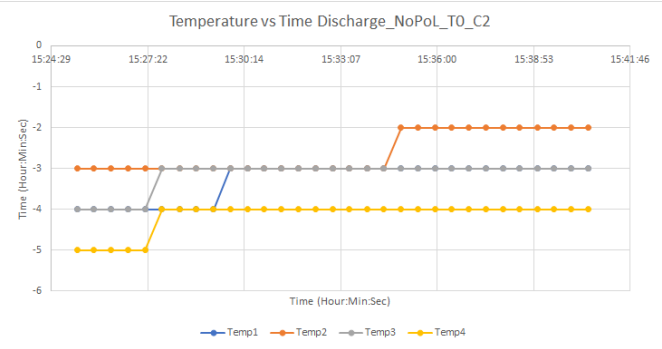
Discharge rate 1st section: 300 mV/h
Discharge rate 2nd section: 141.67 mV/h



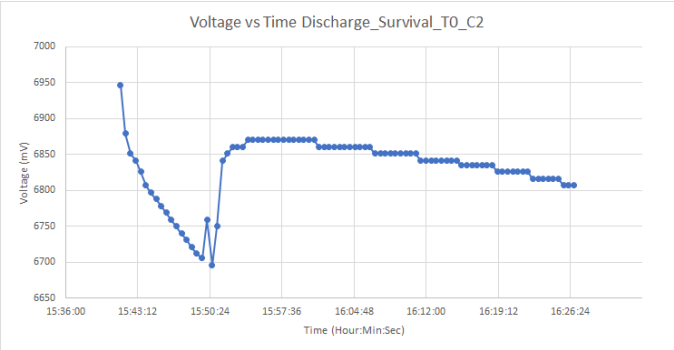
- 2nd Cycle
  - 0ºC Discharge
    - NoPoL

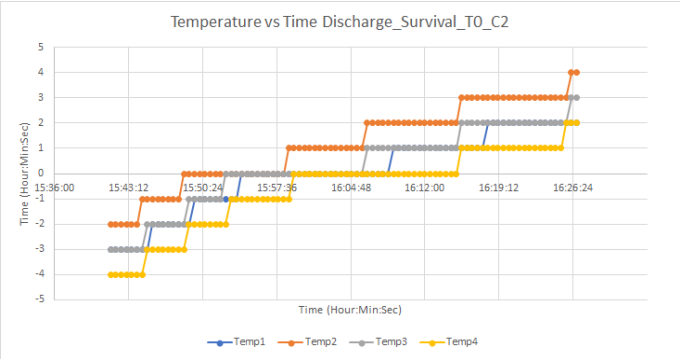



    - Survival

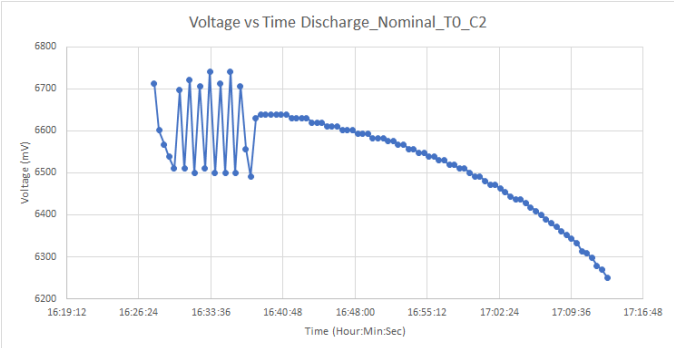
Voltage loss in the 1st section: 245 mV
Discharge rate 2nd section: 83.33 mV/h

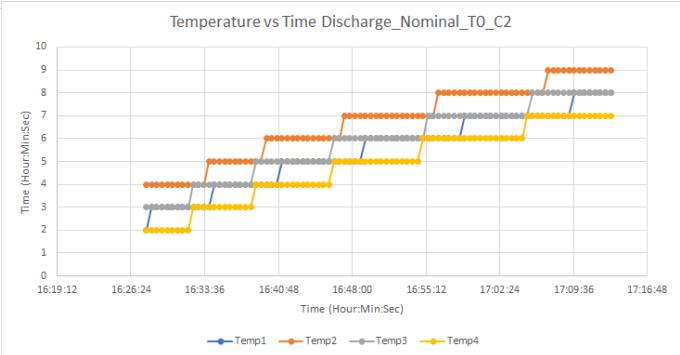



    - Nominal

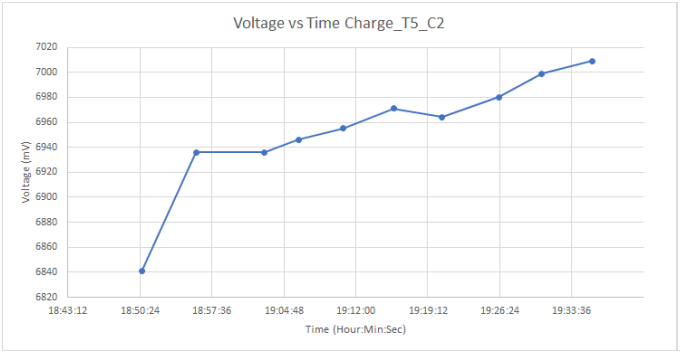
Voltage loss in the 1st section: N/A
Discharge rate 2nd section: 640 mV/h

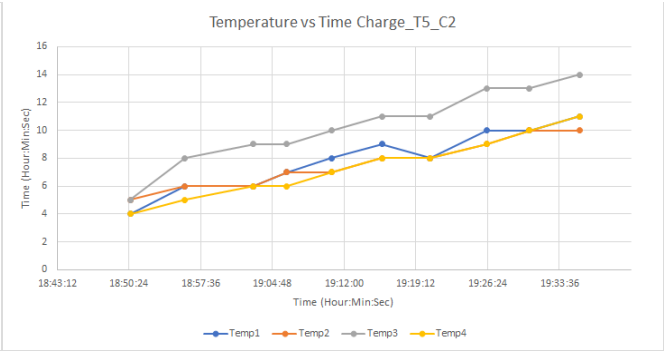


- 5ºC Charge

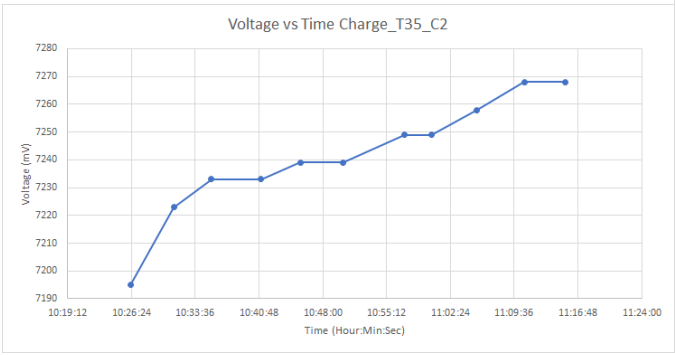
Charge rate : 226.67 mV/h

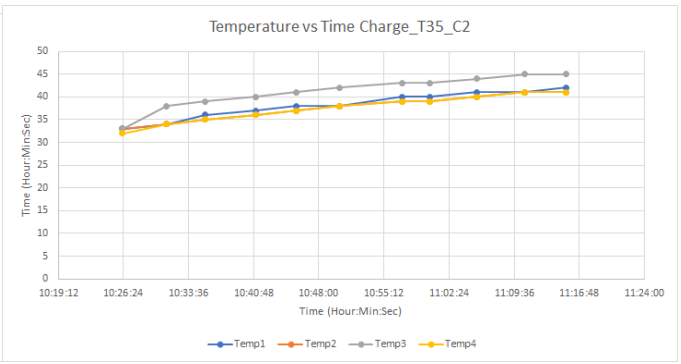


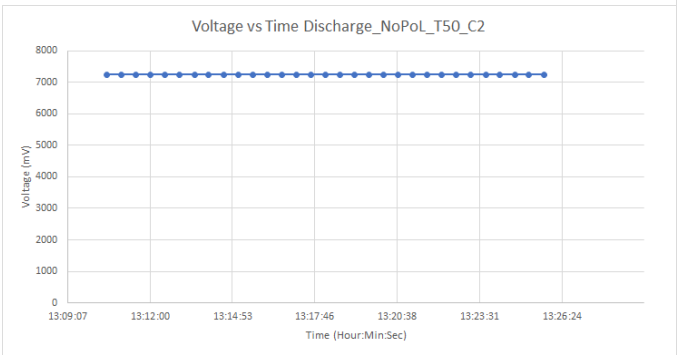

- 35ºC Charge

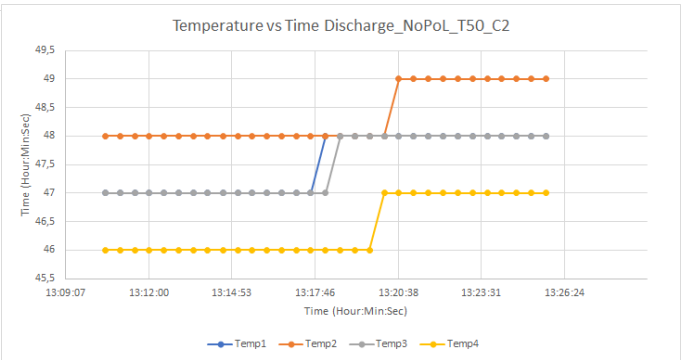
Charge rate : 93.33 mV/h



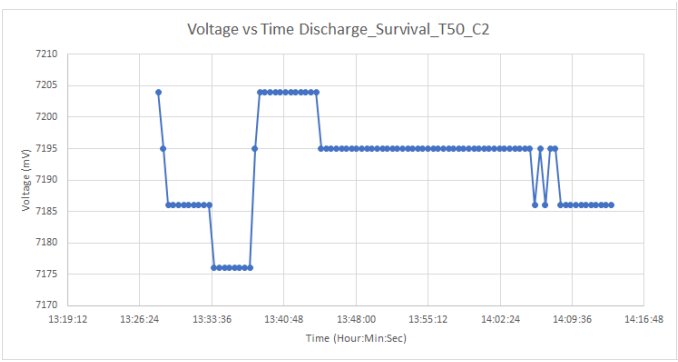

- 50ºC Discharge
  - NoPoL

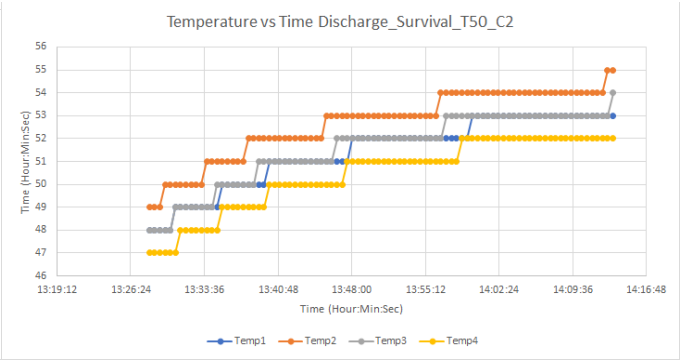


- Survival

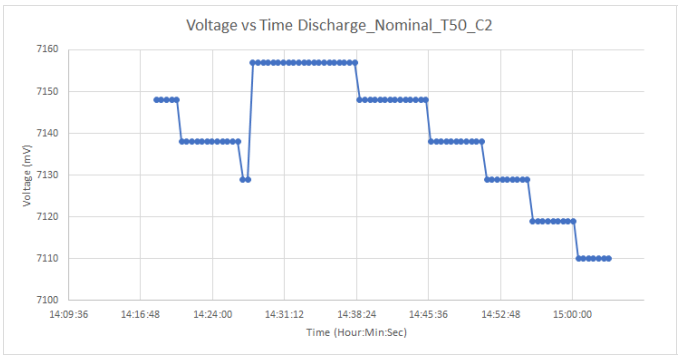
Discharge rate 1st section: 200 mV/h
Discharge rate 2nd section: 33.33 mV/h

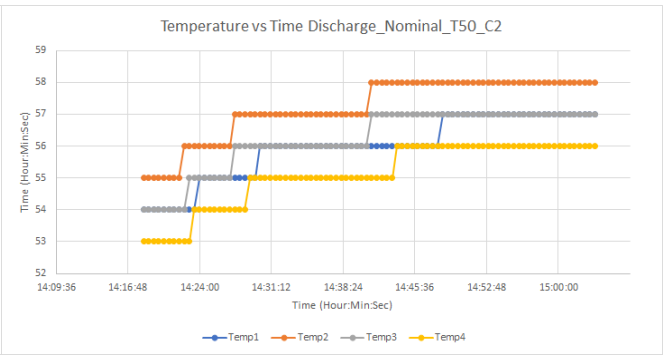


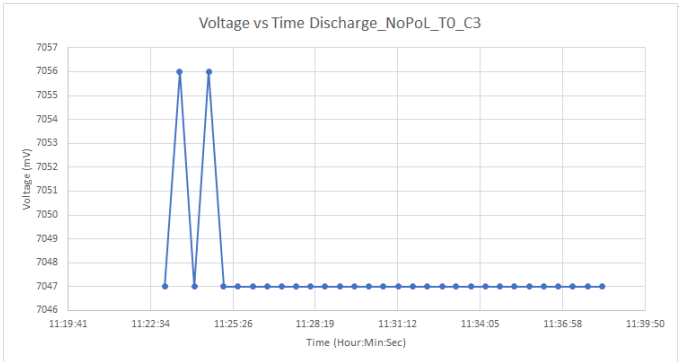

- Nominal

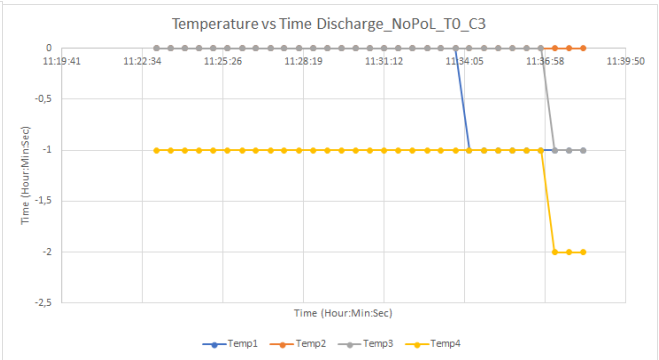
Discharge rate 1st section:133.33 mV/h
Discharge rate 2nd section: 75 mV/h



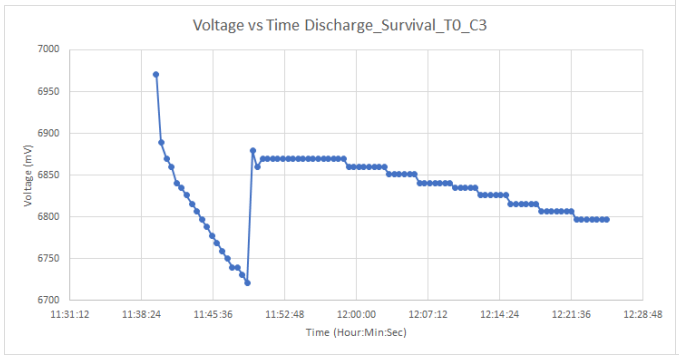

- 3rd Cycle
  - 0ºC Discharge
    - NoPoL

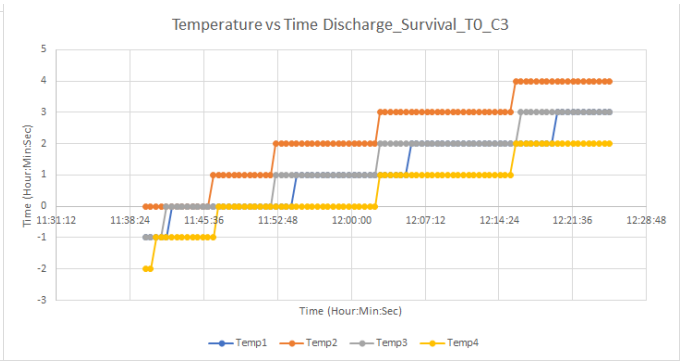


- Survival

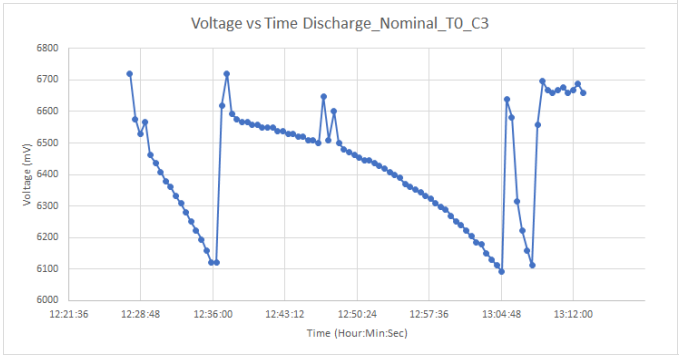
Voltage loss in the 1st section: 250 mV
Discharge rate 2nd section: 116.67 mV/h

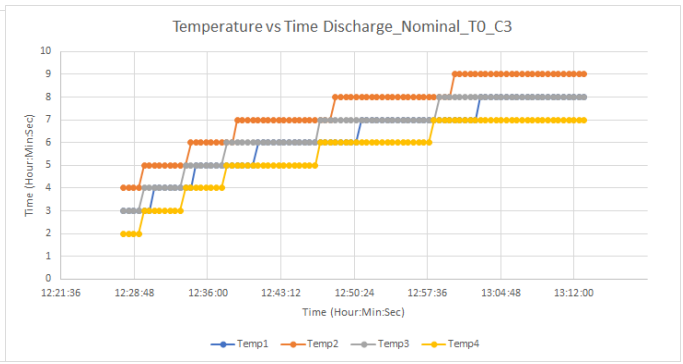



- Nominal

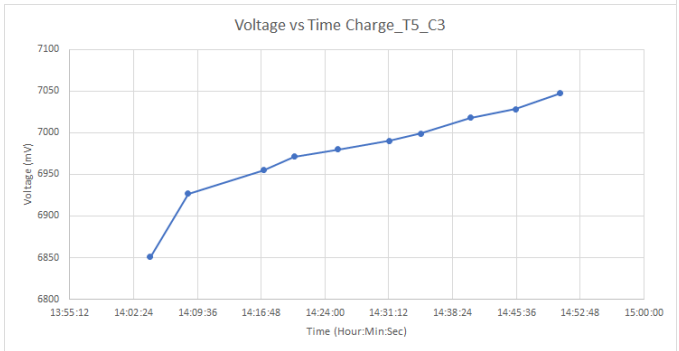
Voltage loss in the 1st section: 600 mV
Discharge rate 2nd section: 966.67 mV/h

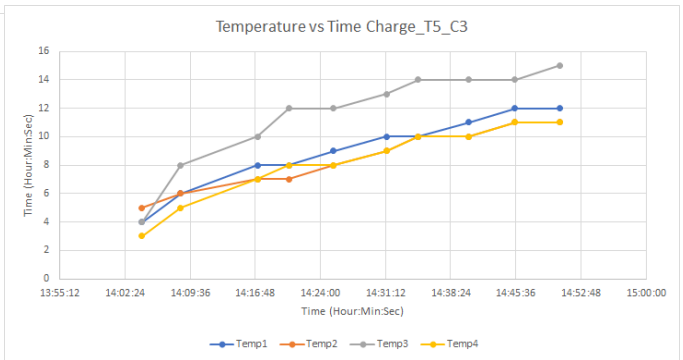



- 5ºC Charge

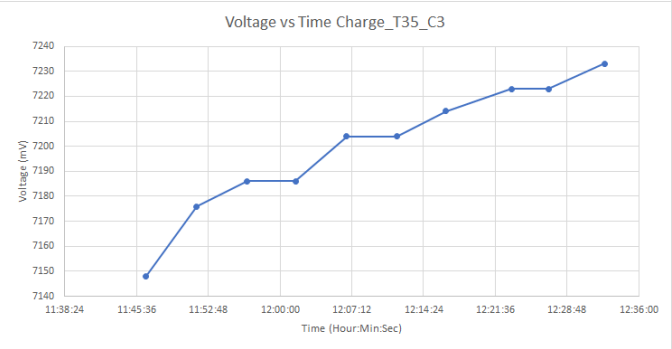
Charge rate : 266.66 mV/h

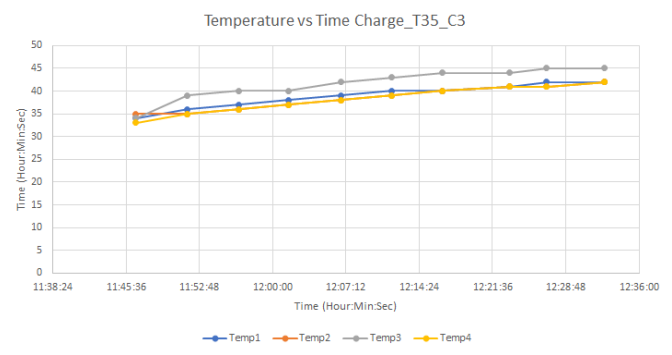


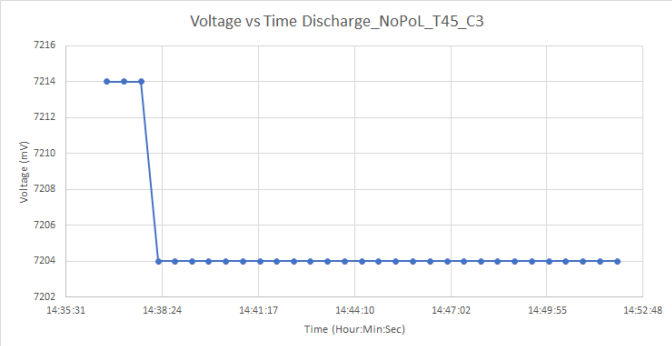
- 35ºC Charge

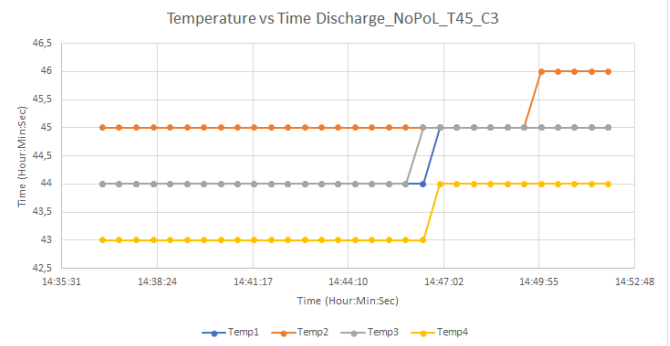
Charge rate : 113.33 mV/h



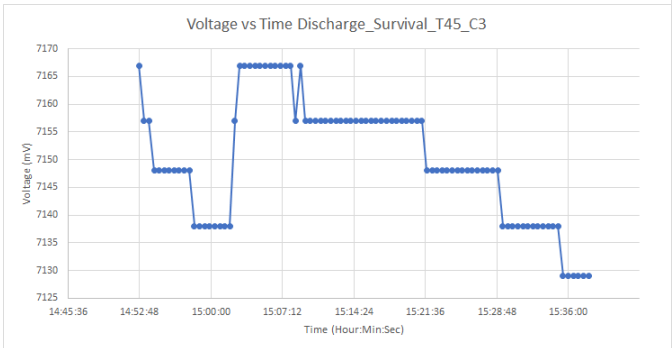

- 45ºC Discharge
  - NoPoL

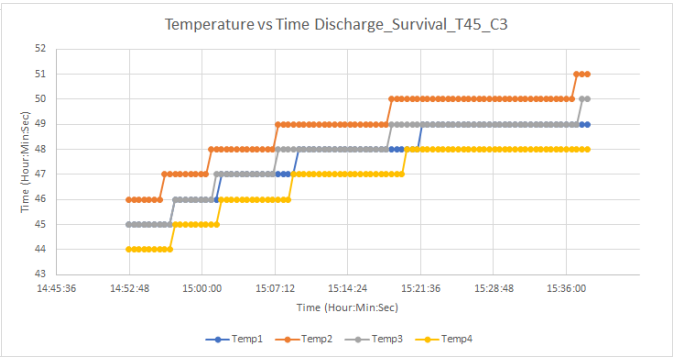



  - Survival

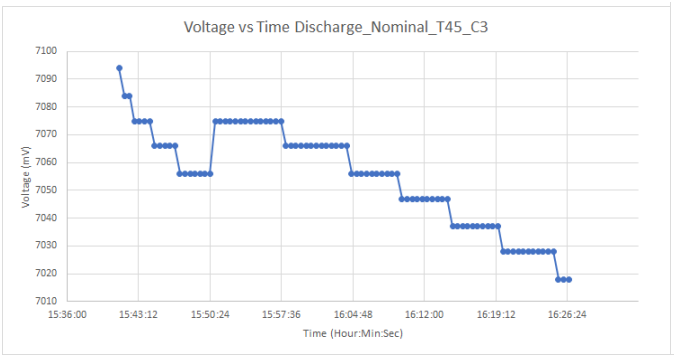
Discharge rate 1st section: 200 mV/h
Discharge rate 2nd section: 58.33 mV/h

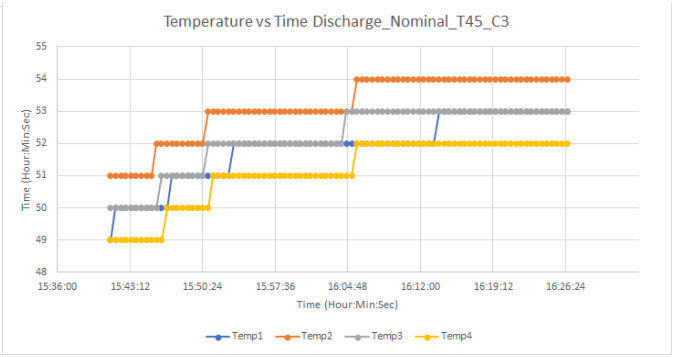


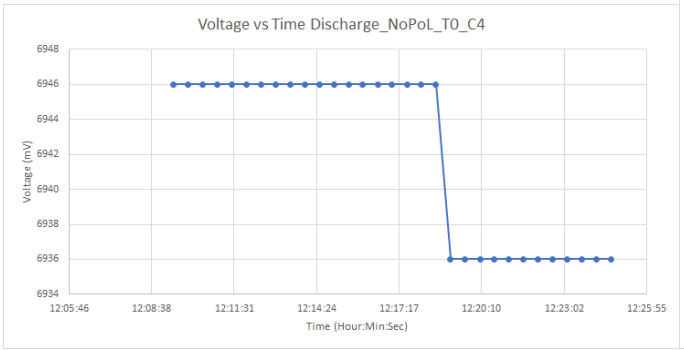
- Nominal

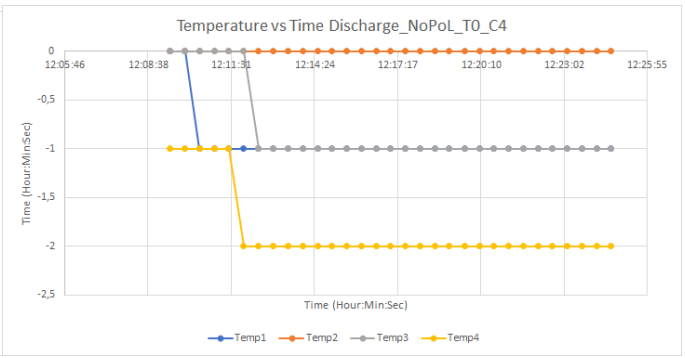
Discharge rate 1st section: 266.67 mV/h
Discharge rate 2nd section: 91.67 mV/h



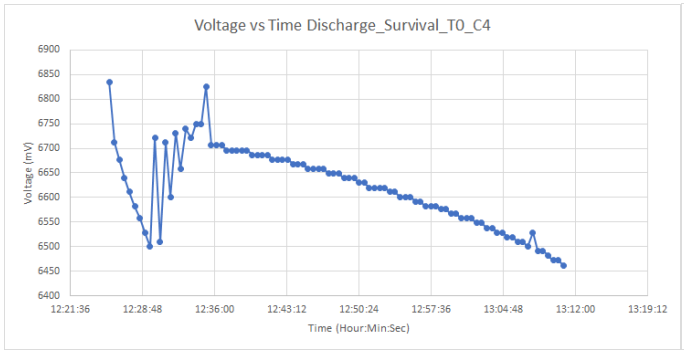

- 4th Cycle
  - 0ºC Discharge
    - NoPoL

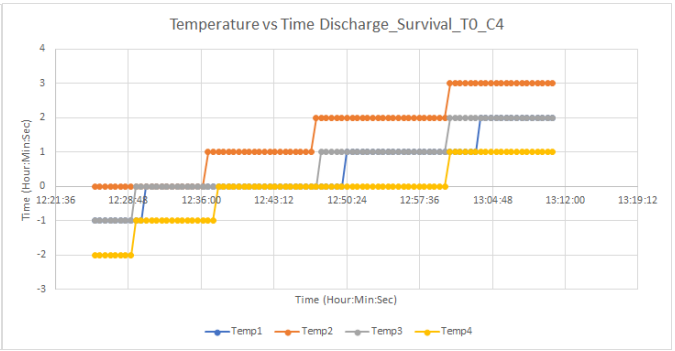



- Survival

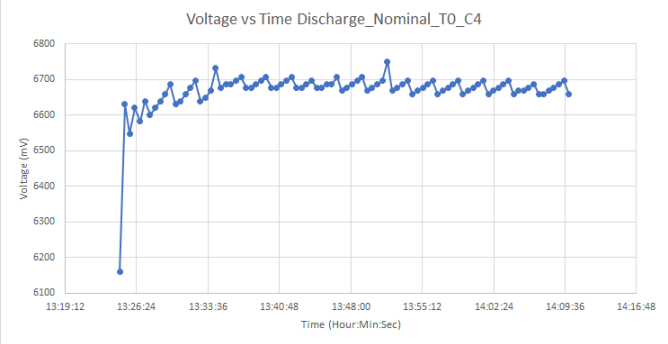
Voltage loss in the 1st section: 350 mV and N/A after 4 minutes
Discharge rate 2nd section: 416.67 mV/h

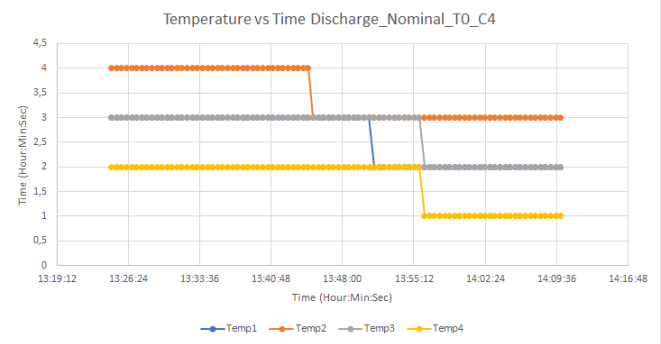


- Nominal

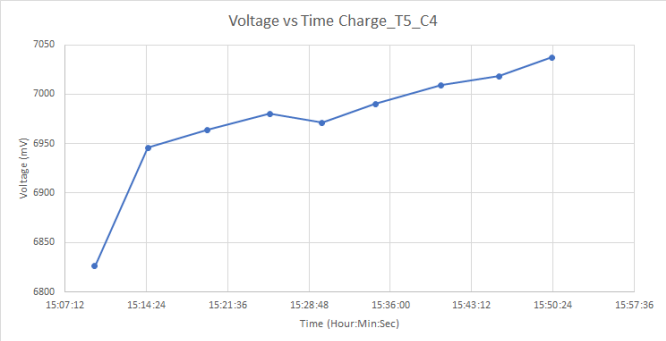
Voltage loss in the 1st section: N/A
Discharge rate 2nd section: N/A

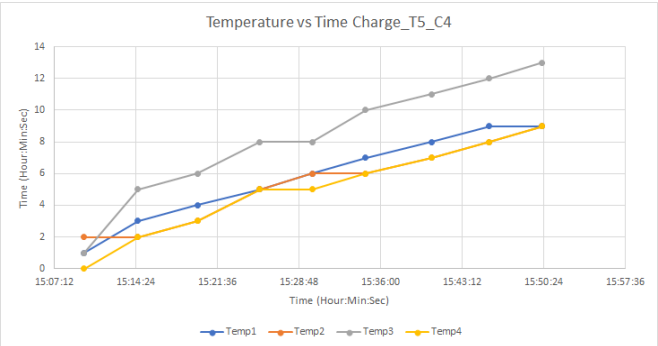



- 5ºC Charge

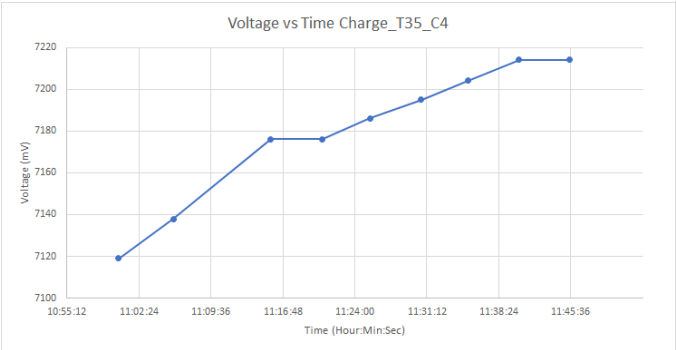
Charge rate : 333.33 mV/h

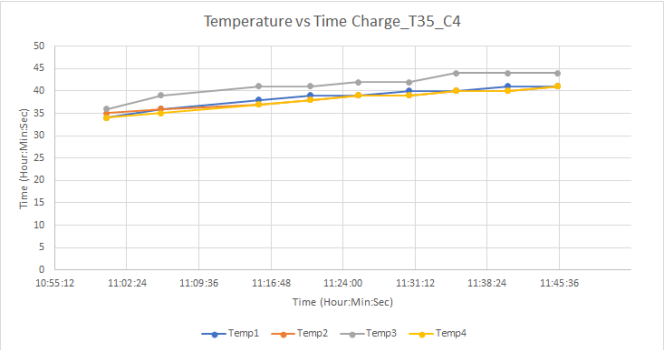


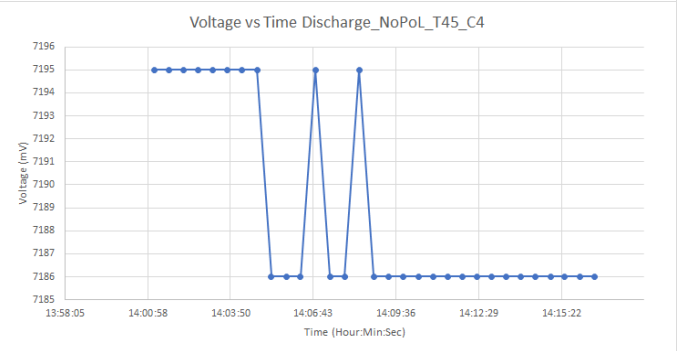

- 35ºC Charge

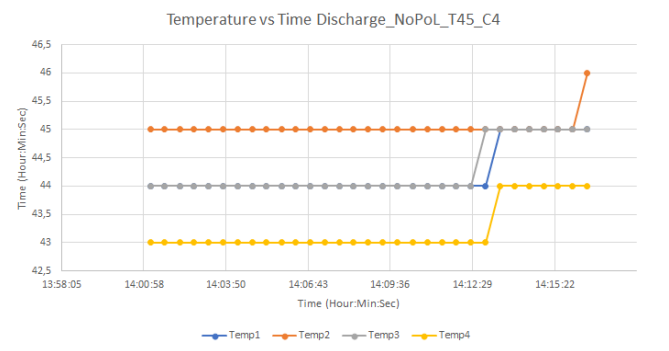
Charge rate : 126.67 mV/h



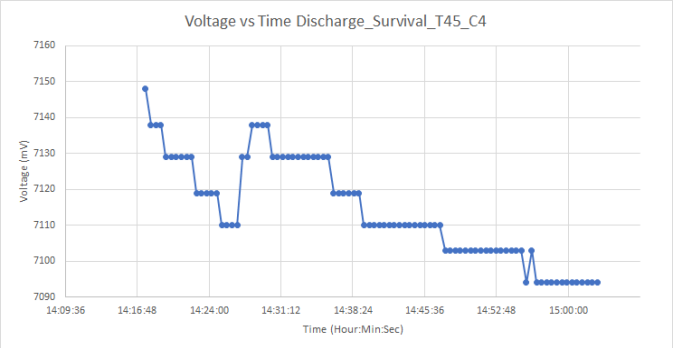
- 45ºC Discharge
  - NoPoL

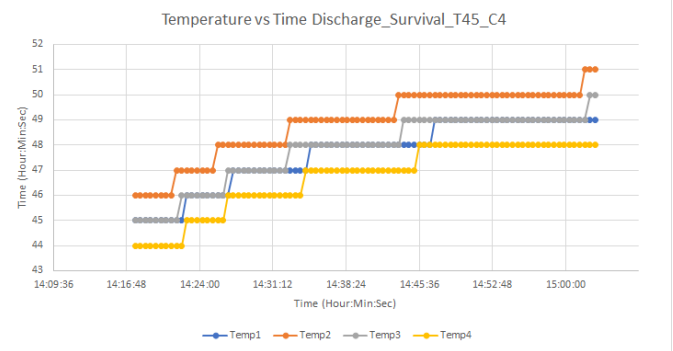



  - Survival

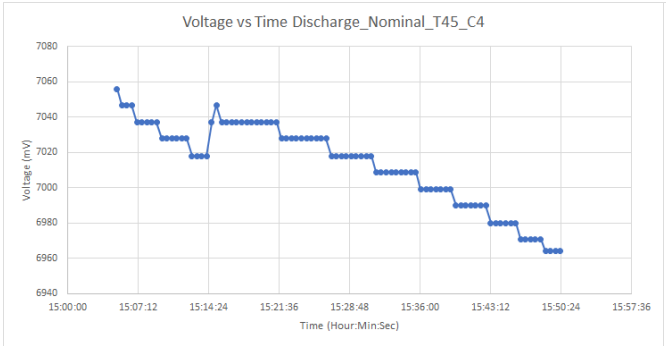
Discharge rate 1st section: 266.67 mV/h
Discharge rate 2nd section: 83.33 mV/h

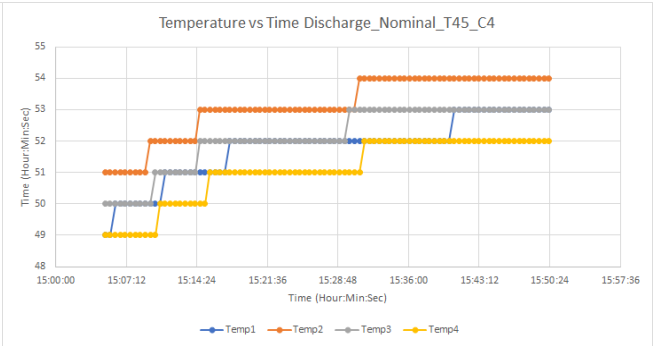



  - Nominal

Discharge rate 1st section: 266.67 mV/h
Discharge rate 2nd section: 125 mV/h




The first thing we want to analyze is the voltage vs temperature plots. We can see how the voltage extremely rises up in both survival and nominal modes 9 minutes after the test has begun. Also, there is a difference in the behaviour of the batteries comparing the cold versus the hot condition. In the cold condition, the voltage drops quicker than in the hot condition.

The charge rate seen in the different plots vary a lot. Normally in the cold temperatures the charge rate is higher than in the hot case, but is certain that the voltage level of the batteries is not the same. In the voltage frame of 7.6V to 8V the charge rate is more or less the same. A characterization of the voltage must be done in order to extract conclusions.
Once the voltage has been analyzed, the temperature vs time will be studied. The plots of temperature show that more or less all the plateaus and different modes follow the same pattern. It is true that the charge mode heats up, but the current is also higher so it was already expected. Within every plateau, the subsystem reaches a temperature of about 10 ºC above the one it started with.

**Synthesis**

Once the analysis has been completed, we will present the hypothesis we have made in order to explain the aforementioned behaviors.

We think that the large hops seen in the voltage vs time plots are due to a loading effect in the voltage measurement. The measurement read is not the actual value of the battery voltage, but the value of the battery voltage lowered by a factor dependent on the resistors.

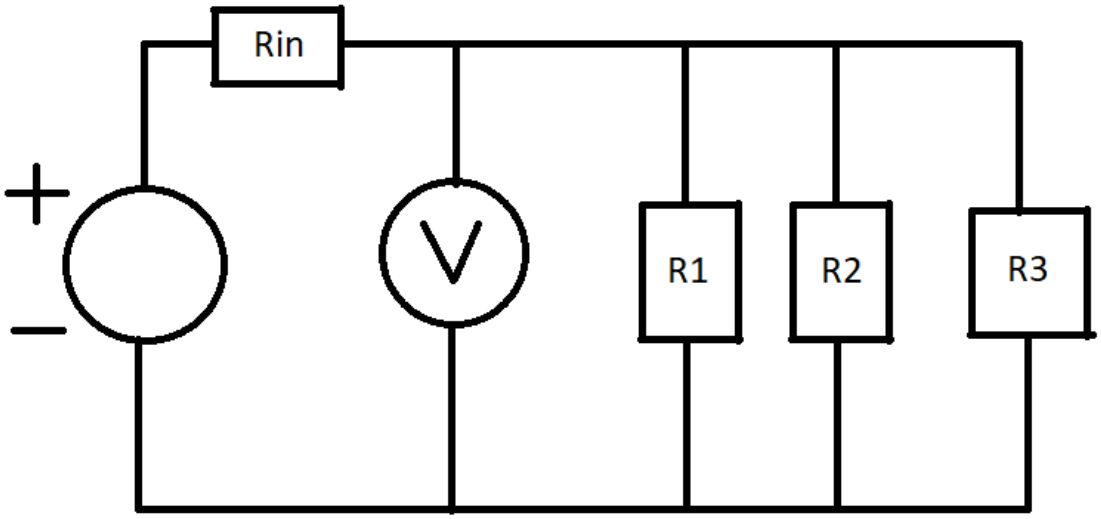



**Figure 7 - Explanation of the batteries (source and Rin) and PoLs, simulated by R1, R2, R3**

The cause of the hops is the inner battery resistance. In order to determine the value and importance of that resistance in our circuit, we have performed some measurements:

- Firstly, we have measured the EPS voltage with the PoL 2 (Magnetorquers) activated, and the voltage obtained has been 6.529 V.

- Then, we have measured the EPS voltage in NoPoL mode (without any PoL active) and we have got a value of 6.582 V.

By doing these two measurements, we are able to determine the value of the inner resistance of the batteries (within this point of charge). The voltage difference is the one that corresponds to the inner resistance, and in this case, the resistance carries a value of 0.203 ohms.

As can be seen in the different plots, the voltage difference caused by this effect fluctuates depending on the EPS level of charge. This is somehow expected because the loading effect is directly proportional to the current voltage. As the voltage increases, the hop will increase in magnitude. This is not the behavior seen as in the low voltage levels the hop can be greater in magnitude than in a higher voltage. Therefore, it can be assumed that the inner resistance changes its value depending on the charge level of the batteries.

So in order to explain the hops, we could say that as the overall resistance increases when the magnetorquers are disconnected (because the PoLs are connected in parallel, so, as the magnetorquers resistance is very low, if it is removed from the parallel resistors, the equivalent resistance increases), the voltage read by the voltmeter will be higher.

The different behaviour in the discharge mode when the subsystem is in a hot or cold condition could be explained since the resistance of the resistors is dependent on temperature. As the resistor increases its temperature, its resistance rises up too.

In the cold condition, the voltage drops quickly because the thermal conduction and radiation properties are poor. Once it starts to heat, the gradient of temperature becomes large as the resistor’s temperature grows fast and the perfboard is near 0°C. Moreover, increasing the temperature improves the thermal radiation properties. This is the point when the temperature of the resistor stabilizes and the voltage discharge rate settles.

In the hot condition the temperature is initially hotter and its thermal equilibrium is closer than it was at 0ºC, so the discharge rate is more stable.

As can be seen in the different plots, the rate of charge is different depending on the EPS charge level. A characterization of the charging rate has been carried out in order to come up with a hypothesis.

Throughout the entire range of possible values that the EPS is able to get, the charge ratios vary.

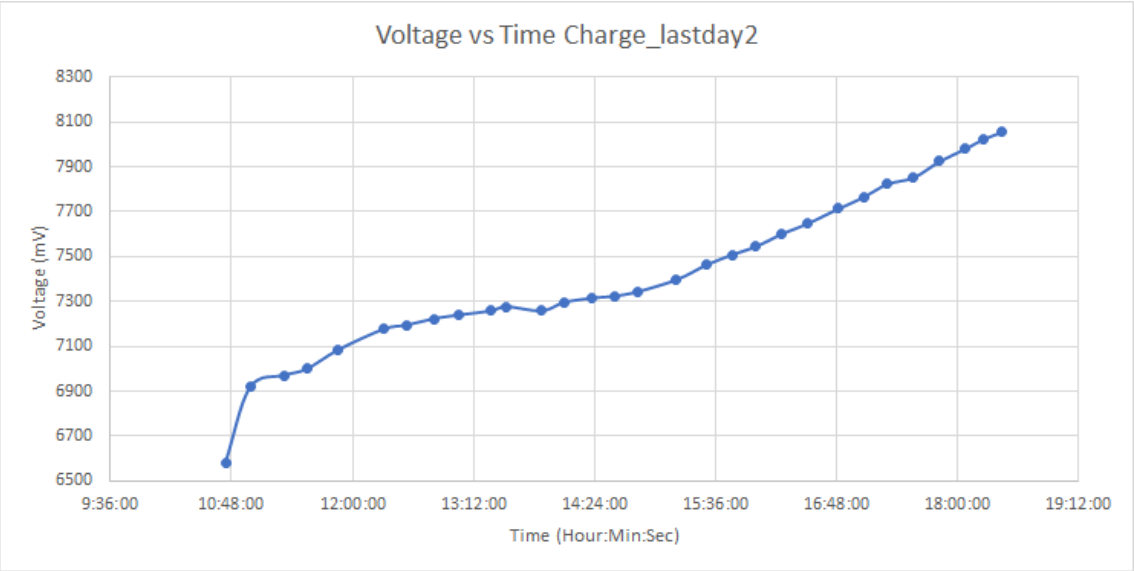



**Figure 8 - Rate of charge throughout the different voltage levels**

The plot states three different charge rates. The first one comprises from 6.5V to 6.9V, the second one, from 6.9V to 7.3V and the third one contains the range from 7.4V to 8.1V.

The first stage is the one with the highest rate of charge, which is 1340 mV/h. During the second stage, the rate of charge decreases, being the smallest value of the 3 stages, 106mV/h. The third stage of the rate of charge includes from 7.4V to approximately the full charge, with a rate of 203 mV/h.